arXiv:2312.16018v2 [cs.IR] 15 Jan 2024

RecRanker: Instruction Tuning Large Language Model as Ranker for Top-k Recommendation

Sichun Luo[1], Bowei He[1], Haohan Zhao[1], Yinya Huang[1], Aojun Zhou[2], Zongpeng Li[3], Yuanzhang Xiao[4], Mingjie Zhan[2], Linqi Song[1]*
[1]City University of Hong Kong [2]The Chinese University of Hong Kong
[3]Hangdian University [4]University of Hawaii
{sichun.luo,boweihe2-c,haohazhao2-c}@my.cityu.edu.hk, {yinya.el.huang,aojunzhou,zmjdll}@gmail.com,
zongpeng@hdu.edu.cn, yxiao8@hawaii.edu, linqi.song@cityu.edu.hk

***Abstract*—Large language models (LLMs) have demonstrated remarkable capabilities and have been extensively deployed across various domains, including recommender systems. Numerous studies have employed specialized *prompts* to harness the in-context learning capabilities intrinsic to LLMs. For example, LLMs are prompted to act as zero-shot rankers for listwise ranking, evaluating candidate items generated by a retrieval model for recommendation. Recent research further use instruction tuning technique to align LLM with human preference for more promising recommendations. Despite its potential, current research overlooks the integration of multiple ranking tasks to enhance model performance. Moreover, the signal from the conventional recommendation model is not integrated into the LLM, limiting the current system performance.**

In this paper, we introduce RecRanker, tailored for instruction tuning LLM to serve as the Ranker for top-*k* Recommendations. Specifically, we introduce importance-aware sampling, clustering-based sampling, and penalty for repetitive sampling for sampling high-quality, representative, and diverse users as training data. To enhance the prompt, we introduce a position shifting strategy to mitigate position bias and augment the prompt with auxiliary information from conventional recommendation models, thereby enriching the contextual understanding of the LLM. Subsequently, we utilize the sampled data to assemble an instruction-tuning dataset with the augmented prompt comprising three distinct ranking tasks: pointwise, pairwise, and listwise rankings. We further propose a hybrid ranking method to enhance the model performance by ensembling these ranking tasks. Our empirical evaluations demonstrate the effectiveness of our proposed RecRanker in both direct and sequential recommendation scenarios.

***Index Terms*—Recommender system, Large language model, Instruction tuning**

I. Introduction

Recommender systems serve as information filtering techniques designed to mitigate the problem of information overload [1]–[4]. Among various scenarios within recommender systems, the top-k recommendation paradigm is particularly noteworthy by providing users with a list of top-k items most relevant to their preferences [5], [6]. Top-k recommendations encompass diverse tasks, including but not limited to, collaborative filtering-based direct recommendations and sequential recommendations. On the one hand, direct recommendations are studied by some prominent methodologies including NCF [7], NGCF [8], and LightGCN [9]. These techniques harness collaborative information via neural networks. On the other hand, for sequential recommendations [10], representative methods like SASRec [11] and BERT4Rec [12] utilize the attention mechanism [13] to model user sequences.

In recent years, large language models (LLMs) [14]–[16] have exhibited significant prowess in natural language understanding [17], generation [18], and complex reasoning [19]. Consequently, they have been increasingly integrated into a multitude of domains, including recommender systems [20]–[22]. A typical example of LLMs in this context is to function as a ranker for a pre-filtered set of recommendations. This preference for LLMs as rankers arises primarily from the inherent limitations of LLMs, including their constrained context size and the potential for high computational costs when processing vast pools of candidate items. Therefore, a retrieval model is often employed to narrow down the candidate set, upon which the LLM utilizes its contextual understanding and reasoning capabilities to generate a ranked list of recommendations. For example, Hou et al. [23] operate LLM as a zero-shot ranker for sequential recommendation by formalizing the recommendation as a conditional ranking task based on sequential interaction histories. By employing carefully designed prompting templates and conducting experiments on standard datasets, they show LLMs exhibit promising zero-shot ranking capabilities that can outperform traditional models. Similar endeavors are also undertaken by [24], [25], where they also leverage the in-context learning abilities of LLMs. However, these methods possess certain limitations. The standard, general-purpose LLM does not inherently align with recommendation objectives.

To remedy this, Zhang et al. [26] suggest employing instruction tuning to better align the LLM with specific recommendation tasks. They express user preferences as natural language instructions, tuning the LLM to deliver more precise and user-centric recommendations. This approach outperforms traditional models and even GPT-3.5 in evaluations. Nonetheless, current research has not provided a thorough study of the ranking task, i.e., most studies deploy LLMs for a singular ranking task, neglecting the exploration of the potential benefits of combining multiple ranking tasks for improved results. Furthermore, prevailing approaches rely

*Corresponding Author

exclusively on textual information of users and items for LLM processing and reasoning. This oversight of not integrating signals from conventional recommendation models may limit the effectiveness of existing methodologies.

To address this shortfall, we introduce instruction tuning large language model as **Ranker** for top-k **Rec**ommendation, referred to as RecRanker. Specifically, we propose an adaptive user sampling method to garner high-quality users, giving priority to users with a substantial history of interactions or who are representative of the broader user base, recognizing their heightened significance in the dataset. To enhance the prompt, we propose position shifting strategy to mitigate position bias. In accordance with the concept of self-consistency in LLM [27], we posit that the answer that receives consensus among most replies is more likely to be accurate. We also incorporate signals from conventional recommendation models into prompts to augment LLM reasoning, as these signals can harness information from broader perspectives. The signals are seamlessly incorporated into the prompt using natural language descriptions in a uniform format. Subsequently, we curate an instruction-tuning dataset with enhanced prompts comprising three distinct ranking tasks, including pointwise, pairwise, and listwise ranking. The instruction tuning dataset is adopted to fine-tune the open-source LLM, resulting in a refined model that is well-aligned with the objectives of recommendation. Furthermore, we introduce a hybrid ranking approach that amalgamates all three ranking methods to bolster model performance. Experiments conducted on three real-world datasets validate the effectiveness of the proposed RecRanker.

In a nutshell, our contribution is fourfold.

- We introduce RecRanker, a compact framework that applies instruction-tuned LLMs for diverse ranking tasks in top-k recommendations. In addition, we propose a hybrid ranking method that ensembles various ranking tasks, aiming to further improve the model performance.
- RecRanker employs adaptive user sampling to select high-quality users, thereby facilitating the construction of the instruction-tuning dataset. Furthermore, we propose a position shifting strategy within the prompt to mitigate the position bias in LLM.
- Our approach incorporates information from conventional recommender systems into the instructions, enabling the LLM to synergistically leverage signals from both the conventional recommender system and textual information for better contextual understanding and user preferences reasoning.
- We conducted extensive experiments on three real-world datasets to validate the effectiveness of our proposed RecRanker. Impressively, RecRanker outperforms backbone models in most cases by a large margin, demonstrating its significant superiority.

II. Related work

A. Top-k Recommendation

Top-k recommendations [5] have emerged as a burgeoning research field, aiming to suggest a list of k items that are most likely to align with a user's preferences. Two predominant categories of algorithms for top-k recommendations are collaborative filtering-based direct recommendation and sequential recommendation. For direct recommendation, memory-based approaches such as user-based and item-based collaborative filtering are employed [28]. These algorithms leverage the historical interactions between users and items to compute similarity scores and then generate recommendations. Advanced methods, including Neural Collaborative Filtering (NCF) [7] and Neural Graph Collaborative Filtering (NGCF) [8], have been developed to better model collaborative user behavior and infer user preferences with more complex model structures. In contrast, sequential recommendation focuses on capturing the dynamic behavior of users. Techniques like Gated Recurrent Unit for Recommendation (GRU4Rec) [29], Self-Attention-based Sequential Recommendation (SASRec) [11], and the more recent transformer-based BERT4Rec [12] utilize the sequential nature of user interactions to predict the forthcoming items of interests to users.

Though conventional algorithms achieve promising results in top-k recommendations, they still lack the ability to understand the content of the items. To address this issue, this paper proposes to facilitate recommender systems by leveraging the contextual understanding and reasoning capabilities of LLMs.

B. LLMs for Recommendation

Recently, LLMs have demonstrated remarkable capabilities and have found extensive applications across various domains, including recommender systems [21], [22]. Some recent works utilize LLMs for data augmentation [30] or representation learning [31]–[33] in recommendations. Notably, one strand of research leverages LLMs as rankers for recommender systems [25], [26]. This approach is necessitated by the limitations of LLMs' fixed window size, which prevents the direct input of an exhaustive set of candidate items. Consequently, a retrieval model is commonly employed to refine and reduce the candidate item set. Specifically, Wang et al. [25] investigated the in-context-learning ability of LLMs with designed task-specific prompts to facilitate ranking tasks in sequential recommendation. However, the misalignment between general-purpose LLMs and specialized recommendation tasks constrains the models' performance. To address this limitation, InstructRec [26] instruction tunes LLMs using a specially constructed dataset of natural language instructions. However, existing research has yet to fully exploit the ranking capabilities of LLMs; it has primarily focused on singular ranking tasks, thereby leaving the ensemble of ranking tasks for improved performance largely unexplored.

To bridge this gap, we conduct a systematic investigation into the application of instruction-tuned LLMs for a variety of ranking tasks, including pointwise, pairwise, listwise, and

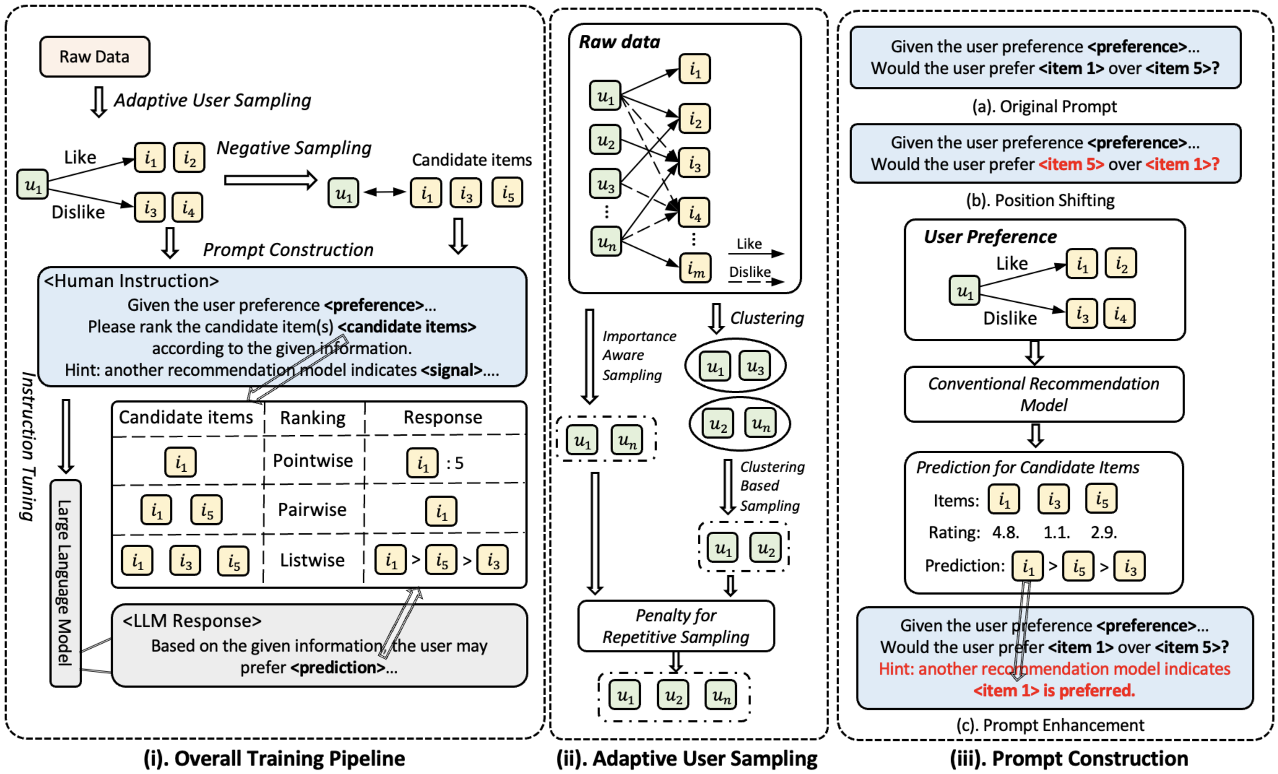


Fig. 1: (i). The overall training pipeline of RecRanker. (ii). Adaptive user sampling module, where we propose importance-aware sampling, clustering-based, and penalty for repetitive sampling to sample users. For each sampled user, corresponding candidate items are randomly selected from the items the user liked, disliked, and has no interaction with. (iii). Prompt construction, where we incorporate position shifting and prompt enhancement strategies to enhance the model performance.

their hybrid approaches, with the objective of fully elucidating the potential of LLMs in top-k recommendation scenarios.

III. PRELIMINARIES

We consider a recommender system with a set of users, denoted $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$, and a set of items, denoted $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$. The top-$k$ recommendation focuses on identifying a subset of items $\mathcal{S}_u \subset \mathcal{I}$ for each user $u \in \mathcal{U}$. The subset is chosen to maximize a user-specific utility $U(u, \mathcal{S})$ with the constraint $|\mathcal{S}| = k$, which is formally expressed as

$$\mathcal{S}_u = \arg\max_{\mathcal{S} \subset \mathcal{I}, |\mathcal{S}|=k} U(u, \mathcal{S}). \quad (1)$$

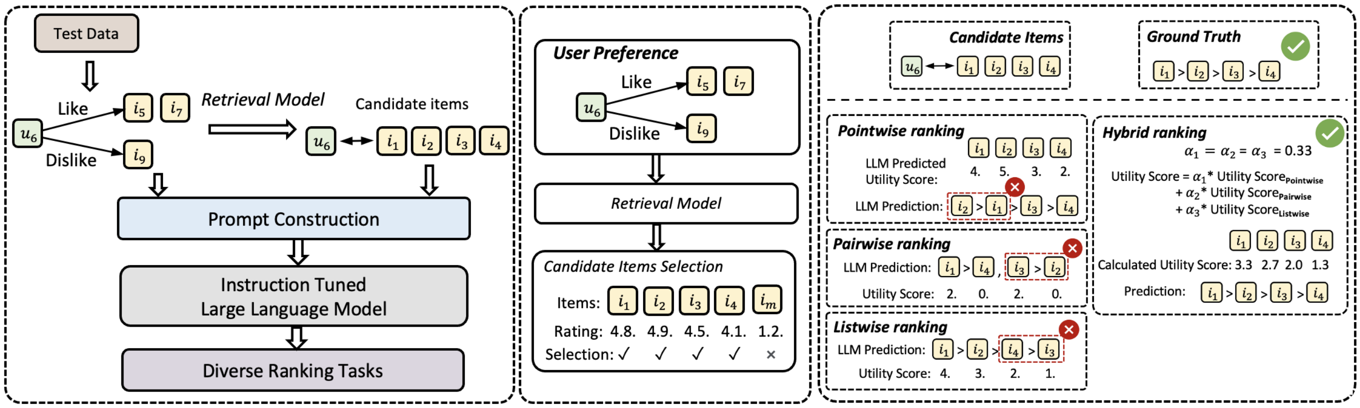
In the context of LLM-based recommendation methods, let $\mathcal{L}$ represent the original LLM. These kinds of methods first utilize *prompts* to interpret the recommendation task for user u into natural language. Given a prompt $\mathcal{P}_u$, the LLM-based recommendation for user u with in-context learning is denoted by $\mathcal{R} = \mathcal{L}(\mathcal{P}_u)$. To fine-tune our LLM using instruction-based approaches, we utilize a dedicated dataset, $\mathcal{D}_{ins}$. The resulting instruction-tuned LLM is represented as $\mathcal{L}'$. Therefore, the recommendation process in the fine-tuned model can be succinctly represented as $\mathcal{R} = \mathcal{L}'(\mathcal{P}_u)$

IV. METHODOLOGY

A. Overview

The overall training and inference pipeline are depicted in Fig. 1 and Fig. 2, respectively. The training phase consists of four main stages: adaptive user sampling, candidate item selection via negative sampling, prompt construction, and instruction tuning. The adaptive user sampling stage aims to procure high-quality, representative, and diverse users. It incorporates three sampling strategies: importance-aware sampling, clustering-based sampling, and penalties for repetition. For each user sampled, the candidate items consist of items liked and disliked by the users, as well as some un-interacted items selected via a commonly used negative sampling method [34], [35]. Given the users sampled and items selected, we construct prompts for each ranking task, augmenting them with signals from conventional recommender models. This strategy synergizes the strengths of both conventional recommendation systems and textual data, thereby enhancing the system's overall performance. Finally, we use the constructed data to fine-tune LLMs via instruction tuning.

During the inference phase, for a user in the test data, we first select candidate items through a retrieval model. This item selection process is different from the training phase, where negative sampling is used. Subsequently, the prompt is



(i). Overall Inference Pipeline (ii). Candidate Items Selection via Retrieval Model (iii). Diverse Ranking Tasks

Fig. 2: (i). The overall inference pipeline of RecRanker. (ii). Candidate item selection via retrieval model, where we adopt the retrieval model to calculate the score for each item and select the highest ones as the candidate items. (iii). Comparison of the proposed hybrid ranking method with three ranking tasks during the inference stage.

constructed, following the approach in the training phase. After that, the instruction-tuned LLM performs a variety of ranking tasks. Notably, a hybrid ranking method, which is achieved through the ensemble of multiple ranking tasks, is employed in this stage to enhance the model performance.

B. Adaptive User Sampling

We first describe how we sample the raw recommendation dataset to create a list of users to be included in the fine-tuning dataset $\mathcal{D}_{ins}$. We do not use the original user set $\mathcal{U}$, because we prefer to generate a list of users with improved distribution and diversity. We denote such a list of users by a *multiset* $\mathcal{U}_{ins}$. A multiset is a modified set that allows for multiple instances of the same element [36]. A multiset is formally defined by a tuple $\mathcal{U}_{ins} = (\underline{\mathcal{U}}_{ins}, M_{ins})$, where $\underline{\mathcal{U}}_{ins}$ is the *underlying set* of the multiset, consisting of its distinct elements, and $M_{ins} : \underline{\mathcal{U}}_{ins} \rightarrow \mathbb{Z}^+$ is the *multiplicity* function, giving the number of occurrences of element $u \in \underline{\mathcal{U}}_{ins}$ as $M_{ins}(u)$. Therefore, the multiplicity $M_{ins}(u)$ of user u will be the number of prompts regarding user u in the instruction-tuning dataset $\mathcal{D}_{ins}$.

Some works sample users with equal probabilities from the user set $\mathcal{U}$ [37], while other works sample nearest interactions [38]. However, these methods could be sub-optimal, since the recommendation dataset often follows a long-tail distribution. To compile a high-quality, representative, and diverse dataset, we introduce three strategies: importance-aware sampling, clustering-based sampling, and penalties for repetitive sampling. Specifically, we utilize importance-aware sampling and clustering-based sampling to create two multisets of candidate users, denoted by $\mathcal{U}_1$ and $\mathcal{U}_2$. Then from the combined multiset $\mathcal{U}_3 = \mathcal{U}_1 + \mathcal{U}_2$ with multiplicity function is $M_3 = M_1 + M_2$, we apply a penalty for repetitive sampling to select the final multiset $\mathcal{U}_{ins}$.

1) Importance-aware Sampling: Data in recommendation scenarios often exhibit a long-tail distribution, where a large number of items or users have minimal interactions, and a few have a large number of interactions [39], [40]. To optimize the quality of the data for building effective recommendation models, we propose an importance-aware sampling strategy. This strategy prioritizes sampling from users with more interactions, based on the premise that users with a higher number of interactions provide more reliable and consistent data, crucial for modeling user preferences accurately. We define the *importance* of a user by the natural logarithm of their interaction count. The importance w_u of user u is defined as $w_u = \ln(q_u)$, where q_u denotes the number of interactions for user u. The logarithmic scale is deliberately chosen to moderate the influence of users with extremely high interaction counts, ensuring that while they are given priority, they do not predominate the entire dataset.

The probability of selecting user u is proportional to the importance w_u. This ensures that users with more interactions have a higher chance of being sampled, while still allowing for representation across the entire user base. In importance-aware sampling, the probability of sampling user u is

$$p_{u,\text{importance}} = \frac{w_u}{\sum_{v \in \mathcal{U}} w_v}, \tag{2}$$

where the denominator is the sum of the importance across all the users, serving as a normalizing factor so that the probabilities sum up to 1.

Importance-aware sampling, as a superior alternative to uniform sampling, offers several advantages. First, it improves data quality by prioritizing users who exhibit a higher volume of interactions, thereby generating a dataset with richer and more consistent patterns. Second, this strategy equitably balances both highly active and less active users by incorporating logarithmic scaling, thereby ensuring that less active users are not underrepresented.

2) Clustering-based Sampling: To obtain representative users, we also employ a clustering-based sampling strategy.

TABLE I: Illustrative examples of instructions for three ranking tasks. For better readability, a modified version of the actual instructions employed in our experiments are shown here.

Type	Instructions
Pointwise Ranking	The historical interactions of a user include: <historical interactions>. How would the user rate <candidate item>?
Pairwise Ranking	The historical interactions of a user include: <historical interactions>. Would the user prefer <candidate item 1>over <candidate item 2>?
Listwise Ranking	The historical interactions of a user include: <historical interactions>. How would the user rank the <candidate item list>?

This strategy is grounded in the understanding that users in recommendation systems exhibit diverse interests. By clustering users in the latent space, we can categorize them into distinct groups, each representing a unique set of interests. Such clustering enables us to capture the multifaceted nature of user preferences, ensuring that our sampling is not only representative but also encompasses the broad spectrum of user behaviors and tendencies.

Our framework allows for any cluster method such as K-means [41] and Mean Shift [42]. In this paper, we choose K-means due to its effectiveness and simplicity in grouping data into cohesive clusters. We first represent each user as an embedding vector derived by the retrieval model, and then cluster the users into K groups based on the embedding vectors. We denote user u's cluster by $k_u \in \{1, \ldots, K\}$. Once the users are clustered, we select samples from each cluster. This selection is not uniform but proportional to the size of each cluster. Mathematically, the sampling probability of user u in clustering-based sampling satisfies

$$p_{u,\text{clustering}} \propto \left|\{v \in \mathcal{U} : \; k_v = k_u\}\right|, \tag{3}$$

where $|\{v \in \mathcal{U} : \; k_v = k_u\}|$ is the number of users in the same cluster as user u. This strategy not only preserves the diversity within each cluster but also ensures that larger clusters, which potentially represent more prevalent interests, have a proportionally larger representation in the final sample.

3) Penalty for Repetitive Sampling: Given the two multisets $\mathcal{U}_1$ and $\mathcal{U}_2$ resulting from the importance-aware and clustering-based samplings, we need to construct the final user list $\mathcal{U}_{ins}$ from their sum $\mathcal{U}_3 = \mathcal{U}_1 + \mathcal{U}_2$, where the multiplicity function is $M_3 = M_1 + M_2$.

To enhance diversity in the final multiset $\mathcal{U}_{ins}$, we implement a penalty for repetitive selections. The rationale behind this strategy is to mitigate the overrepresentation of certain "advantage groups" — users or items that might dominate the dataset due to their high frequency or popularity [39], [40]. To achieve this, we assign a penalty weight for each repeated selection within our sampling process. The penalty weight for a user $u \in \mathcal{U}_3$ is quantitatively expressed as $\psi_u = C^{M_3(u)}$, where $0<C<1$ is a predefined constant. Thus, the penalty weight is decreasing in the number of occurrences $M_3(u)$. This penalty weight directly influences the probability of a user being selected for the final dataset. To be specific, the probability of selecting user u is

$$p_{u,\text{penalty}} = \frac{\psi_u}{\sum_{v \in \mathcal{U}_3} \psi_v}, \tag{4}$$

which ensures that those with higher occurrences are less likely to be chosen repeatedly.

This penalty for repetitiveness serves a dual purpose. Firstly, it significantly enhances the diversity of the sample by reducing the likelihood of repeatedly selecting the same users. Secondly, it ensures a more equitable representation of less frequent users, providing a more holistic view of user interests and preferences. In this way, by integrating this penalty mechanism into our sampling process, we achieve diversity and balanced representation in the final user list $\mathcal{U}_{ins}$.

C. Candidate Items Selection

The selection of candidate items differs between the training and inference phases. During training, negative sampling is utilized to select a mixture of items with which users have not interacted, as well as a random assortment of items that users have liked or disliked, forming the set of candidate items. While in the inference phase, a retrieval model is employed to generate the entire set of candidate items.

1) Selection via Negative Sampling in The Training Phase: In the training phase, the candidate item set includes randomly chosen items that users have liked and disliked. Besides, we employ the widely-used negative sampling technique [34], [35], [43], which involves randomly incorporating items with which users have not interacted into the candidate item set. These un-interacted items are considered as *negative samples*. It is presumed that un-interacted items are more likely to be preferred over items that users have explicitly disliked. Based on these selections, we establish the relative ranking comparison for the instruction tuning dataset construction.

2) Selection via Retrieval Model in The Inference Phase: In the realm of industrial recommender systems, platforms like YouTube[1] often adopt a two-step process, initially utilizing a retrieval model to select a preliminary set of candidate items, which are subsequently re-ranked for final recommendations [44]. Specifically, within LLM-based recommendation systems, the retrieval model plays a crucial role as a primary filter, effectively narrowing the scope of potential recommendations. This is particularly important due to the intrinsic limitations in the window size of LLMs. The architecture of the retrieval model is tailored to suit the nature of the recommendation task at hand. For direct recommendation, models such as NCF [7], NGCF [8], and LightGCN [9] are often employed. For sequential recommendation tasks, where the order of interactions is significant, models like SASRec [11] and BERT4Rec [12] are typically favored.

In the procedure of candidate item selection in the inference phase, we employ the retrieval model to compute a utility score for each item. Subsequently, we rank all the items based on

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

their utility scores and select the top k' items with the highest scores as the candidate items. For top-k recommendations, this process will sample k' items with $k'>k$.

D. Prompt Construction

In this section, we describe the construction of prompts. We begin by introducing a variety of ranking tasks, followed by a discussion of our proposed prompt enhancement method. This method involves augmenting prompts with signals from a conventional recommendation model.

1) Pointwise, Pairwise, and Listwise Ranking: Our recommendation system incorporates a multifaceted approach to ranking tasks, encompassing pointwise, pairwise, and listwise rankings. Each of these methods plays a distinct role in evaluating and ordering candidate items based on their relevance to user preferences. As demonstrated in Table I, for pointwise ranking approach, each candidate item is assigned an individual relevance score. The entire list of candidates is then sorted based on these scores, providing a straightforward, score-based ranking. The pairwise ranking method involves a direct comparison between two candidate items, determining which of the two is more relevant or preferable in a given context. Differing from the above two, listwise ranking evaluates and sorts an entire list of candidate items. It considers the collective relevance of items, offering a comprehensive ranking based on overall suitability.

2) Position Shifting in Prompt: Position bias in LLMs arises when these models disproportionately favor items due to their locations in a list, rather than their inherent relevance or quality [45], [46]. This bias can significantly undermine the consistency and reliability of the output of the model. To mitigate the position bias, we adopt a position shifting strategy. During the training phase, we randomize the order of candidates and user preference items. This strategy is designed to prevent the model from prioritizing the item position over its actual significance. Similarly, in the inference phase, we continue this strategy by randomly altering the positions of the items. The primary objective of this strategy is to preserve those responses from LLMs that demonstrate consistency irrespective of item position. Consequently, the items identified are reflective of the model's true preferences, less influenced by position bias. By employing this method, we ensure that the LLMs' responses are founded on genuine relevance, thereby enhancing the overall trustworthiness of the inference process.

3) Prompt Enhancement: Existing LLM-based approaches often rely solely on LLMs for processing and ranking textual information. This reliance, however, neglects the rich and valuable signals that conventional recommendation models, like collaborative filtering, can offer. Models such as LightGCN [9] excel in extracting high-order collaborative signals, which play a pivotal role in understanding user preferences through the influences of user networks. The absence of the collaborative information could lead to less effective outcomes in LLM-based recommendations.

To bridge this gap, we propose a prompt enhancement method that integrates signals from conventional recommendation models into the prompts used for ranking tasks. This integration allows us to leverage the strengths of both LLMs and traditional recommendation models, creating a more informed and context-rich basis for decision-making. Specifically, for pointwise ranking, we could utilize a rating prediction model like MF [47] to forecast individual scores. These predictions are then transformed into natural language descriptions and seamlessly integrated into the prompt, providing a more nuanced basis for item evaluation. For pairwise and listwise rankings, task-specific models such as LightGCN [9] and SASRec [11] are employed to predict rankings. In this paper, we adopt MF [47] and the LightGCN [9] model for prompt enhancement. The insights from these predictions are then incorporated into the prompts, enhancing the context and depth of the ranking process. By augmenting prompts with data from conventional recommendation models, our method significantly enriches the ranking tasks in recommendation systems. This innovative approach not only capitalizes on the advanced capabilities of LLMs but also harnesses the collaborative or sequential information offered by conventional recommendation models.

E. Optimization via Instruction Tuning

After constructing the dataset, we focus on fine-tuning the LLM in a supervised manner, specifically through instruction tuning. This process involves optimizing the LLM using a dataset generated from instructional data, aligning the model responses more closely with user intents and preferences.

The approach we adopt for supervised fine-tuning is grounded in the standard cross-entropy loss, following the principles outlined in Alpaca [48]. The core of this process lies in the training set $\mathcal{D}_{ins}$, which is comprised of natural language instruction input-output pairs (x, y). This dataset is instrumental in guiding the fine-tuning process, ensuring that the model outputs are aligned with the structured instructional data.

The primary objective in this phase is to fine-tune the pre-trained LLM $\mathcal{L}$ by minimizing the cross-entropy loss. This is mathematically formalized as:

$$\min_{\Theta} \sum_{(x,y)\in\mathcal{D}_{ins}} \sum_{t=1}^{|y|} - \log P_{\Theta}\left(y_t \mid x, y_{[1:t-1]}\right), \qquad (5)$$

where Θ represents the model parameters, P_{Θ} denotes the conditional probability of generating the t-th token y_t in the target output y, given the input x and the preceding tokens $y_{[1:t-1]}$, and $|y|$ is the length of the target sequence y.

By minimizing this loss function, the model parameters Θ are refined to better accommodate the nuances of the instructional tuning dataset $\mathcal{D}_{ins}$. This fine-tuning leverages the LLM's pre-existing capabilities in general language understanding and reasoning, as acquired during its initial training phase. The result is a more sophisticated and nuanced model that can accurately capture and interpret user preferences expressed in natural language. Such an enhancement is crucial for the subsequent recommendation tasks, as it allows the

LLM to provide recommendations that are more aligned with the user's expressed needs and preferences. This approach, therefore, significantly boosts the efficacy and relevance of the recommendation system, ensuring that it serves users with high accuracy and personalization.

F. Hybrid Ranking

Inspired by self-consistency in LLM [27], the result agreed by most LLM responses has a higher probability of being correct. Recognizing that each ranking task (i.e., pointwise, pairwise, and listwise ranking) captures different facets of the recommendation problem, we propose a hybrid ranking method. This method aims to amalgamate the strengths of each individual task to achieve a more holistic and effective recommendation process. The hybrid ranking method operates by ensembling the outputs of the three distinct ranking tasks. Mathematically, this process can be expressed as:

$$\mathcal{U} = \alpha_1 \mathcal{U}_{pointwise} + \alpha_2 \mathcal{U}_{pairwise} + \alpha_3 \mathcal{U}_{listwise} \tag{6}$$

where α_1, α_2, and α_3 are weighting coefficients that sum up to 1. Depending on the values of these coefficients, the hybrid ranking can effectively mimic any of the individual ranking methods, thus providing flexibility in the recommendation approach. For the pointwise ranking task, the utility score, $\mathcal{U}_{pointwise}$, is initially determined by the relevance score from the LLM prediction. To refine this score and differentiate between items with identical ratings, an additional utility score from the retrieval model is incorporated, denoted as $\mathcal{U}_{retrieval} = -m \cdot \mathcal{C}_1$. Here, $\mathcal{C}_1$ is a constant and m, representing the item's position as determined by the retrieval model, varies from 1 to k' (total number of candidate items). Therefore, the comprehensive utility score for the pointwise ranking task is $\mathcal{U}_{pointwise} = \mathcal{U}_{retrieval} + \mathcal{L}(\mathcal{P})$. In the pairwise ranking scenario, preferred items by LLM are attributed a utility score $\mathcal{U}_{pairwise} = \mathcal{C}_2$, where $\mathcal{C}_2$ is a constant. For listwise ranking, the formula $\mathcal{U}_{listwise} = -m' \cdot \mathcal{C}_3$ is employed to score each item, with m' is the position predicted by LLM and varying from 1 to k' and $\mathcal{C}_3$ being a constant. This formula assigns scores across the list of items, integrating the listwise perspective into the hybrid approach.

V. EXPERIMENT

TABLE II: Dataset Description.

Dataset	# of User	# of Item	# of Rating	Density
ML-100K	943	1,682	100,000	0.063046
ML-1M	6,040	3,706	1,000,209	0.044683
BookCrossing	77,805	185,973	433,671	0.000030

The primary goal is to investigate the extent to which integrating the introduced model can improve the performance of current recommendation systems. Therefore, we conduct comprehensive experiments to answer the following research questions:

- **RQ1**: Does our proposed RecRanker framework enhance the performance of existing recommendation models?
- **RQ2**: What impact do importance aware sampling and enhanced prompt have on the quality of recommendation respectively?
- **RQ3**: How do various hyper-parameters influence the overall performance of the framework?
- **RQ4**: How does the instruction-tuned model compare to other LLMs, such as GPT?

A. Experimental Setup

1) Dataset: Following [38], we rigorously evaluate the performance of our proposed framework by employing three heterogeneous, real-world datasets. **MovieLens**[2] [49] dataset is utilized as a standard benchmark in movie recommendation systems. We explore two subsets of this dataset: MovieLens-100K, containing 100,000 user-item ratings, and MovieLens-1M, which expands to approximately 1 million ratings. **BookCrossing**[3] [50] dataset comprises user-submitted book ratings on a 1 to 10 scale and includes metadata such as 'Book-Author' and 'Book-Title'. The key statistics of these datasets are detailed in Table II.

2) Evaluation Metrics: In line with the methodologies adopted in prior works [9], [12], we employ two well-established metrics for evaluating the top-k recommendation task: Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG), denoted as H and N, respectively. Our experimental setup involves setting k to either 3 or 5, similar to the evaluation approach detailed in [26], allowing for a comprehensive assessment.

3) Data Preprocessing: To assure data quality in our study, we implement the 10-core setting, which involves excluding users and items that have fewer than ten interactions from the BookCrossing dataset. The processed BookCrossing dataset, configured with a 10-core setting, comprises 1,820 users, 2,030 items, and 41,456 interactions, resulting in a density of 0.011220. We adopt the leave-one-out evaluation strategy, aligning with the methodologies employed in prior research [26], [51]. Under this strategy, the most recent interaction of each user is assigned as the test instance, the penultimate interaction is used for validation, and all preceding interactions constitute the training set. Regarding the construction of the instruction-tuning dataset, we sampled 10,000 instructions for each ranking task for the ML-1M dataset. In the case of the ML-100K and BookCrossing datasets, we formulated 5,000 instructions for each task, respectively. We eliminated instructions that were repetitive or of low quality (identified by users with fewer than three interactions in their interaction history), leaving approximately 56,000 high-quality instructions. These instructions are then combined to create a comprehensive instruction-tuning dataset, which is utilized to fine-tune the LLM.

4) Model Selection: We incorporate our RecRanker with the following direct recommendation models as the backbone models:

[2]https://grouplens.org/datasets/movielens/

[3]In the absence of timestamp data within the BookCrossing dataset, we have reconstructed historical interactions via random sampling.

TABLE III: Performance achieved by different direct recommendation methods. The best results are highlighted in **boldfaces**.

Backbone	Method	ML-100K				ML-1M				BookCrossing			
		H@3 ↑	N@3 ↑	H@5 ↑	N@5 ↑	H@3 ↑	N@3 ↑	H@5 ↑	N@5 ↑	H@3 ↑	N@3 ↑	H@5 ↑	N@5 ↑
MF	Base	0.0455	0.0325	0.0690	0.0420	0.0255	0.0187	0.0403	0.0248	0.0503	0.0389	0.0689	0.0465
	RecRanker$_{pointwise}$	0.0660	0.0486	0.0917	0.0592	0.0294	0.0213	0.0456	0.0279	0.0872	0.0710	**0.0966**	0.0749
	RecRanker$_{pairwise}$	0.0533	0.0368	0.0783	0.0471	0.0275	0.0201	0.0438	0.0268	0.0539	0.0419	0.0716	0.0492
	RecRanker$_{listwise}$	0.0464	0.0346	0.0712	0.0448	0.0271	0.0196	0.0416	0.0256	0.0430	0.0312	0.0674	0.0411
	RecRanker$_{hybrid}$	**0.0690**	**0.0513**	**0.0919**	**0.0607**	**0.0312**	**0.0230**	**0.0469**	**0.0294**	**0.0873**	**0.0720**	**0.0966**	**0.0759**
	Improvement	51.65%	57.85%	33.19%	44.52%	22.35%	22.99%	16.38%	18.55%	73.56%	85.09%	40.20%	63.23%
LightGCN	Base	0.0492	0.0343	0.0744	0.0447	0.0273	0.0197	0.0431	0.0261	0.0645	0.0499	0.0875	0.0595
	RecRanker$_{pointwise}$	0.0723	0.0524	**0.0990**	**0.0634**	**0.0324**	**0.0232**	0.0480	0.0296	0.1076	0.0876	**0.1231**	0.0940
	RecRanker$_{pairwise}$	0.0414	0.0298	0.0645	0.0393	0.0287	0.0205	0.0450	0.0272	0.0622	0.0481	0.0840	0.0572
	RecRanker$_{listwise}$	0.0509	0.0380	0.0764	0.0485	0.0281	0.0202	0.0440	0.0267	0.0566	0.0395	0.0854	0.0513
	RecRanker$_{hybrid}$	**0.0731**	**0.0527**	0.0971	0.0625	0.0320	**0.0232**	**0.0497**	**0.0305**	**0.1088**	**0.0888**	0.1219	**0.0942**
	Improvement	48.58%	53.64%	33.06%	41.83%	18.68%	17.77%	15.31%	16.86%	68.68%	77.96%	40.69%	58.32%
MixGCF	Base	0.0537	0.0412	0.0736	0.0492	0.0144	0.0108	0.0232	0.0144	0.0746	0.0584	0.0957	0.0671
	RecRanker$_{pointwise}$	0.0701	0.0542	0.0930	0.0637	0.0170	0.0126	0.0263	0.0164	**0.1113**	0.0916	0.1208	0.0955
	RecRanker$_{pairwise}$	0.0537	0.0413	0.0770	0.0508	0.0169	0.0123	0.0265	0.0163	0.0686	0.0556	0.0885	0.0638
	RecRanker$_{listwise}$	0.0507	0.0376	0.0738	0.0470	0.0154	0.0111	0.0252	0.0151	0.0558	0.0415	0.0823	0.0523
	RecRanker$_{hybrid}$	**0.0712**	**0.0551**	**0.0932**	**0.0641**	**0.0180**	**0.0133**	**0.0269**	**0.0169**	**0.1113**	**0.0918**	**0.1209**	**0.0958**
	Improvement	32.59%	33.74%	26.63%	30.28%	25.00%	23.15%	15.95%	17.36%	49.20%	57.19%	26.33%	42.77%
SGL	Base	0.0505	0.0380	0.0729	0.0472	0.0284	0.0206	0.0434	0.0267	0.0609	0.0476	0.0812	0.0560
	RecRanker$_{pointwise}$	**0.0693**	0.0517	**0.0885**	0.0596	0.0320	0.0230	0.0492	0.0301	**0.0951**	**0.0793**	0.1044	**0.0831**
	RecRanker$_{pairwise}$	0.0470	0.0349	0.0710	0.0447	0.0292	0.0211	0.0451	0.0275	0.0590	0.0468	0.0785	0.0549
	RecRanker$_{listwise}$	0.0535	0.0385	0.0740	0.0470	0.0286	0.0207	0.0436	0.0269	0.0493	0.0362	0.0760	0.0471
	RecRanker$_{hybrid}$	0.0690	**0.0525**	0.0882	**0.0604**	**0.0325**	**0.0235**	**0.0497**	**0.0305**	0.0950	0.0791	**0.1045**	**0.0831**
	Improvement	37.23%	38.16%	21.40%	27.97%	14.44%	14.08%	14.52%	14.23%	56.16%	66.60%	28.69%	48.39%

TABLE IV: Performance achieved by different sequential recommendation methods. The best results are highlighted in **boldfaces**.

Backbone	Method	ML-100K				ML-1M				BookCrossing			
		H@3 ↑	N@3 ↑	H@5 ↑	N@5 ↑	H@3 ↑	N@3 ↑	H@5 ↑	N@5 ↑	H@3 ↑	N@3 ↑	H@5 ↑	N@5 ↑
SASRec	Base	0.0187	0.0125	0.0385	0.0205	0.0277	0.0165	0.0501	0.0257	0.0150	0.0086	0.0279	0.0139
	RecRanker$_{pointwise}$	0.0206	0.0147	0.0432	0.0239	**0.0308**	0.0206	**0.0541**	0.0301	0.0351	0.0247	0.0482	0.0302
	RecRanker$_{pairwise}$	**0.0277**	**0.0190**	**0.0479**	**0.0273**	0.0289	0.0195	0.0502	0.0282	0.0227	0.0149	0.0357	0.0203
	RecRanker$_{listwise}$	0.0204	0.0149	0.0321	0.0197	0.0239	0.0162	0.0407	0.0231	0.0218	0.0153	0.0323	0.0196
	RecRanker$_{hybrid}$	0.0232	0.0160	0.0436	0.0243	0.0304	**0.0212**	0.0526	**0.0303**	**0.0381**	**0.0270**	**0.0487**	**0.0315**
	Improvement	48.13%	52.00%	24.42%	33.17%	11.19%	28.48%	7.98%	17.90%	154.00%	213.95%	74.55%	126.62%
BERT4Rec	Base	0.0153	0.0104	0.0294	0.0161	0.0107	0.0069	0.0211	0.0112	0.0179	0.0119	0.0343	0.0185
	RecRanker$_{pointwise}$	0.0183	0.0129	0.0334	0.0191	**0.0140**	**0.0095**	**0.0231**	**0.0133**	0.0390	0.0279	0.0557	0.0348
	RecRanker$_{pairwise}$	**0.0194**	**0.0133**	0.0334	0.0190	0.0090	0.0061	0.0159	0.0089	0.0254	0.0173	0.0416	0.0240
	RecRanker$_{listwise}$	0.0162	0.0119	0.0240	0.0151	0.0124	0.0085	**0.0231**	0.0128	0.0242	0.0168	0.0371	0.0221
	RecRanker$_{hybrid}$	0.0191	0.0130	**0.0343**	**0.0192**	0.0135	0.0094	0.0230	**0.0133**	**0.0422**	**0.0305**	**0.0566**	**0.0365**
	Improvement	26.80%	27.88%	16.67%	19.25%	30.84%	37.68%	9.48%	18.75%	135.75%	156.30%	65.01%	97.30%
CL4SRec	Base	**0.0243**	0.0143	**0.0436**	0.0222	0.0259	0.0153	0.0492	0.0248	0.0151	0.0088	0.0282	0.0141
	RecRanker$_{pointwise}$	0.0219	0.0149	0.0417	**0.0229**	**0.0290**	**0.0196**	**0.0519**	**0.0289**	0.0349	0.0241	0.0501	0.0304
	RecRanker$_{pairwise}$	0.0226	**0.0155**	0.0404	**0.0229**	0.0278	0.0186	0.0462	0.0261	0.0190	0.0132	0.0361	0.0202
	RecRanker$_{listwise}$	0.0200	0.0145	0.0323	0.0195	0.0232	0.0156	0.0413	0.0230	0.0219	0.0152	0.0323	0.0194
	RecRanker$_{hybrid}$	0.0221	0.0152	0.0400	0.0224	0.0280	0.0192	0.0512	0.0286	**0.0375**	**0.0263**	**0.0514**	**0.0321**
	Improvement	N/A	8.39%	N/A	3.15%	11.97%	28.10%	5.49%	16.53%	148.34%	198.86%	82.27%	127.66%

- **Matrix Factorization (MF)** [47]: A foundational approach that decomposes user-item interaction matrices to uncover latent features. We use Bayesian Personalized Ranking (BPR) loss [34] to optimize the model.
- **LightGCN** [9]: Simplifies the graph convolutional network for efficient recommendation by focusing on user-item graph embeddings.
- **MixGCF** [52]: A hybrid method combining graph convolution with collaborative filtering, enhancing recommendation diversity and accuracy.
- **SGL** [53]: Utilizes self-supervised learning within graph neural networks to improve recommendation quality through auxiliary tasks.

We also employ several widely used sequential recommendation models as the backbones.

- **SASRec** [11]: Employs a self-attention mechanism in sequential models to better capture user preferences over time.
- **BERT4Rec** [12]: Adapts the BERT architecture to sequential recommendation, capturing complex item interaction patterns.
- **CL4SRec** [54]: Leverages contrastive learning for sequential recommendation, enhancing model robustness and understanding of user-item sequences.

The backbone models serve as the retrieval models in RecRanker. For each backbone model, we choose the top ten items as candidate items, setting $k' = 10$.

We leave out the comparison with other instruction-tuning LLM for recommendation methods such as TALLRec [38] and

InstructRec [37]. This exclusion is justified as these methods are not primarily designed for diverse ranking tasks. Specifically, TALLRec is tailored for a binary classification task, determining whether a user likes an item or not. InstructRec, on the other hand, relies on the powerful yet closed-source GPT model to generate information, rendering it impractical in our context. Nevertheless, it is important to note that these methods adhere to the standard approach for instruction tuning in LLMs. As detailed in Section V-C, we include an ablation study that evaluates our method's enhancements over the standard instruction tuning LLMs, thereby underscoring the superiority of our approach.

5) Implementation Details: We chose LLaMA-2 (7B) [15] as the backbone of LLM in our experiment due to its strong capability among the open-source LLMs. In the training phase of LLaMA-2 (7B), we adopted a uniform learning rate of 2×10^{-5} , coupled with a context length of 1024. The batch size was fixed at 4, complemented by gradient accumulation steps of 2. Additionally, a cosine scheduler was implemented, integrating a preliminary warm-up phase of 50 steps. The training comprised a total of 6000 steps. We employed DeepSpeed's ZeRO-3 stage optimization [55] alongside the flash attention technique [56] for efficient training of these models. This training process was executed on 16 NVIDIA A800 80GB GPUs. During the inference process, the vLLM framework [57] was employed, setting the temperature parameter at 0.1, with top-k and top-p values at 10 and 0.1, respectively. Inference was conducted using a single NVIDIA A800 80GB GPU.

For the top-k recommendation task, we utilize the SELFRec[4] library [53] for implementation. As for the hyper-parameter settings, we set $\alpha_1 = \alpha_2 = \alpha_3 = \frac{1}{3}$ for all experiments. $\mathcal{C}$ is set to 0.92 in this paper. $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ are set to 0.05, 0.5, and 0.025 respectively. We repeat the experiment five times and calculate the average.

B. Main Results (RQ1)

The experiment results for direct recommendation and sequential recommendation are shown in Table III and Table IV respectively. We have the following key observations:

- In the context of MF and LightGCN, pairwise and listwise ranking methods surpass the baseline model. However, these methods encounter difficulties in yielding favorable outcomes when applied to more advanced models like MixGCF or SGL. In contrast, pointwise ranking consistently outperforms the base models, achieving a marked improvement. This enhancement might be attributed to the LLM proficiency in making more objective judgments, rather than comparing multiple items. Additionally, the relative simplicity of pointwise tasks suggests that LLMs are more adept at handling simpler tasks.
- Furthermore, hybrid ranking methods generally outperform pointwise ranking. Despite the significantly lower performance of pairwise and listwise ranking compared to pointwise ranking, integrating them into a hybrid ranking approach can still result in improvements. This is in line with the concept of self-consistency in LLMs; that is, when a model consistently agrees on a particular answer, there is a higher likelihood of its accuracy.
- RecRanker demonstrates a more significant improvement on the Bookcrossing dataset than on the Movielens dataset. This enhancement may be due to the fine-grained ratings in Bookcrossing dataset, which range from 1 to 10, thereby enabling the tuned LLM to make more precise predictions.

[4]https://github.com/Coder-Yu/SELFRec

TABLE V: Ablation Study on ML-100K dataset with backbone model MF for pairwise ranking. The best results are highlighted in boldfaces.

Variants	H@3 ↑	N@3 ↑	H@5 ↑	N@5 ↑
RecRanker	**0.0533**	**0.0368**	**0.0783**	**0.0471**
w/o Adaptive User Sampling	0.0472	0.0347	0.0759	0.0465
w/o Position Shifting	0.0472	0.0337	0.0764	0.0456
w/o Prompt Enhancement	0.0494	0.0358	0.0742	0.0459

This observation can be attributed to the fact that the general recommendation models have the capability to mine collaborative information effectively, which makes them more excel at ranking items. As a result, the need for reranking is comparatively lower in these models.

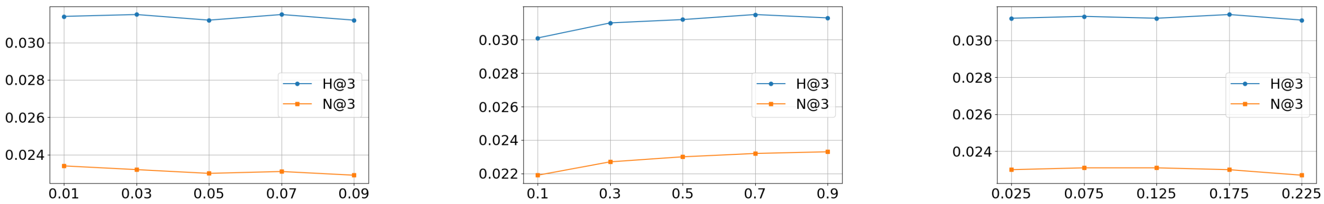
C. Ablation Study (RQ2)

In this section, we study the benefits of each individual component of ReRanker. The results are demonstrated in Table V. The results demonstrate that the complete model outperforms all three model variants. This outcome underscores the significant contribution of each main component to the enhancement of overall performance. A detailed analysis of each component's specific impact yielded the following insights:

- ***w/o* Adaptive User Sampling**: This variant substitutes the proposed adaptive user sampling with a uniform sampling approach. The experimental results reveal a notable decline in model performance. This decline underscores the importance of adaptive user sampling in selecting critical, representative, and diverse user samples for training, thereby enhancing model performance.
- ***w/o* Position Shifting**: The position shifting is excluded in this variant, maintaining other components the same. The observed performance reduction in this variant highlights the significance of position shifting. It mitigates position bias, leading to more consistent and reliable results.
- ***w/o* Prompt Enhancement**: In this variant, prompt enhancement is removed while retaining other modules. A marked decrease in performance is observed, suggesting that conventional recommender models may provide valuable information for LLM to generate more accurate predictions.

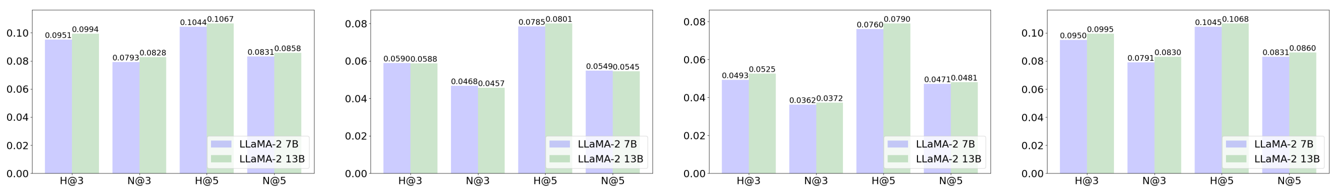
D. Hyper-parameter Study (RQ3)

1) Analysis of hyper-parameters $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$: We analyze the influence of hyper-parameters $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ on the



(a) Impact of $\mathcal{C}_1$ (b) Impact of $\mathcal{C}_2$ (c) Impact of $\mathcal{C}_3$

Fig. 3: Analysis of hyper-parameters $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ on ML-1M dataset with backbone model MF and hybrid ranking task.



(a) Pointwise Ranking (b) Pairwise Ranking (c) Listwise Ranking (d) Hybrid Ranking

Fig. 4: Performance comparison for LLaMA-2 (7B) and LLaMA-2 (13B) model with respect to different ranking tasks on the Bookcrossing dataset with backbone model SGL.

ML-1M dataset, employing MF as the underlying model, as depicted in Figure 3. We noted that increases in $\mathcal{C}_1$ and $\mathcal{C}_3$ led to fluctuations and a general decline in performance. This indicates that judicious selection of $\mathcal{C}_1$ and $\mathcal{C}_3$ is crucial for optimizing model performance, particularly since both pairwise and listwise ranking methods underperform compared to pointwise ranking, rendering high values of $\mathcal{C}_1$ and $\mathcal{C}_3$ suboptimal. On the other hand, a gradual improvement in performance was observed with the increment of $\mathcal{C}_2$. These findings underscore the significance of appropriate hyper-parameter selection in achieving optimal model performance.

2) Analysis of model scaling.: We further instruction-tuned the LLaMA-2 (13B) model.[5] We conducted a comparative analysis between the 7B and 13B versions of the instruction-tuned models. The performance differences between LLaMA-2 7B and LLaMA-2 13B were specifically assessed across various ranking tasks within the Bookcrossing dataset, as illustrated in Figure 4. Our observations revealed that the LLaMA-2 (13B) model generally outperformed the 7B model. This superiority can be attributed to the enhanced capabilities of the larger model, which result in better language comprehension and reasoning ability, ultimately leading to improved ranking outcomes. In addition, It is noteworthy that the improvements in pointwise ranking and listwise ranking were more pronounced compared to pairwise ranking. This suggests that LLMs still face challenges in certain ranking tasks. Furthermore, the hybrid ranking approach demonstrated significant progress across all evaluation metrics. This underscores the effectiveness of integrating multiple ranking tasks, highlighting the strengths of the proposed hybrid ranking method.

[5]Training the LLaMA-2 (70B) model with the same experimental settings was impractical due to resource constraints, consistently resulting in Out-Of-Memory (OOM) errors.

TABLE VI: Performance comparison w.r.t different numbers of instructions for training RecRanker on the ML-100K dataset using the MF backbone model for pairwise ranking

# of Instructions	H@3 ↑	N@3 ↑	H@5 ↑	N@5 ↑
56K	**0.0533**	**0.0368**	**0.0783**	**0.0471**
28K	0.0481	0.0348	0.0757	0.0462
5.6K	0.0475	0.0353	0.0723	0.0454

3) Analysis of data scaling.: The training of the LLM was conducted with varying quantities of instructions in the instruction-tuning dataset to evaluate the effect of data size. Specifically, the version with 5.6K instructions was trained over 600 steps, while the version with 28K instructions underwent 3000 steps of training, proportional to our original configuration. The experiment result is detailed in Table VI. An observable trend is that an increase in the number of instructions correlates with enhanced model performance. This underscores the significance of incorporating a larger and more diverse dataset for instruction tuning LLMs to achieve improved performance.

E. Comparison with the GPT Model (RQ4)

We compare our instruction-tuned LLM with the GPT model, specifically, the GPT-3.5-turbo[6] model. We employed a sample of 100 listwise ranking task instances from the Bookcrossing dataset, using the CLSRec model as the backbone for evaluating the GPT model. This experiment setting aligns with the findings of [58], which highlight the optimal

[6]https://platform.openai.com/docs/models/gpt-3-5

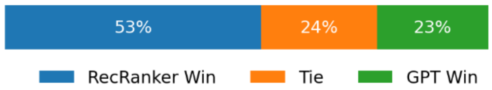


Fig. 5: Comparison between our instruction-tuned model with the GPT-3.5-turbo model.

cost-performance equilibrium achieved when GPT-3.5 is applied to the listwise ranking task. As demonstrated in Figure 5, our instruction-tuned RecRanker with hybrid ranking notably outperforms the GPT-3.5 model. This impressive result emphasizes the crucial role of instruction tuning in aligning general-purpose LLMs specifically for recommendation tasks.

F. Further Discussion

In our experiment, we observed that training the LLaMA-2 7B model with around 56K instructions on 16 A800 GPUs took approximately 4.6 hours. Besides, training the LLaMA-2 13B model under the same conditions required around 5.3 hours. The inference time for each instruction averaged about 17 instructions per second, translating to a requirement of around 0.059 seconds per item for computation by a single A800 GPU.

This training and inference duration significantly exceeds that of conventional recommendation models, highlighting the limitations of current LLM-based recommender systems. The substantial demand for computational resources also represents a significant challenge. Consequently, employing instruction LLMs for large-scale industrial recommender systems, such as those with millions of users, is presently impractical. However, future advancements in accelerated and parallel computing algorithms for language model inference could potentially reduce inference times and computation resources. This improvement might make the integration of LLMs into large-scale recommender systems feasible, especially by leveraging many GPUs for parallel computation.

VI. Conclusion

In this paper, we introduce RecRanker, a novel framework for employing instruction tuning LLM as the **Ranker** in top-k **Rec**ommendations. Initially, we propose an adaptive user sampling for obtaining high-quality, representative, and diverse data. In the following step, we construct an instruction-tuning dataset that encompasses three distinct ranking tasks: pointwise, pairwise, and listwise rankings. We further improve the prompt by adopting position shifting strategy to mitigate position bias, as well as integrating auxiliary information from conventional recommendation models for prompt enhancement. Moreover, we introduce a hybrid ranking method that combines these diverse ranking tasks to improve overall model performance. Extensive empirical studies on three real-world datasets across diverse rankings tasks validate the effectiveness of our proposed framework.

References

[1] B. He, X. He, Y. Zhang, R. Tang, and C. Ma, "Dynamically expandable graph convolution for streaming recommendation," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 1457–1467.
[2] B. He, X. He, R. Zhang, Y. Zhang, R. Tang, and C. Ma, "Dynamic embedding size search with minimum regret for streaming recommender system," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023, pp. 741–750.
[3] S. Luo, Y. Xiao, and L. Song, "Personalized federated recommendation via joint representation learning, user clustering, and model adaptation," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 4289–4293.
[4] S. Luo, Y. Xiao, X. Zhang, Y. Liu, W. Ding, and L. Song, "Perfedrec++: Enhancing personalized federated recommendation with self-supervised pre-training," *arXiv preprint arXiv:2305.06622*, 2023.
[5] X. Yang, H. Steck, Y. Guo, and Y. Liu, "On top-k recommendation using social networks," in *Proceedings of the sixth ACM conference on Recommender systems*, 2012, pp. 67–74.
[6] Z. Zhu, J. Wang, and J. Caverlee, "Improving top-k recommendation via jointcollaborative autoencoders," in *The World Wide Web Conference*, 2019, pp. 3483–3482.
[7] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.
[8] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, 2019, pp. 165–174.
[9] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. 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*, 2020, pp. 639–648.
[10] X.-H. Chen, B. He, Y. Yu, Q. Li, Z. Qin, W. Shang, J. Ye, and C. Ma, "Sim2rec: A simulator-based decision-making approach to optimize real-world long-term user engagement in sequential recommender systems," *arXiv preprint arXiv:2305.04832*, 2023.
[11] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *2018 IEEE international conference on data mining (ICDM)*. IEEE, 2018, pp. 197–206.
[12] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 1441–1450.
[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[14] OpenAI, "Gpt-4 technical report," 2023.
[15] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.
[16] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.
[17] M. Du, F. He, N. Zou, D. Tao, and X. Hu, "Shortcut learning of large language models in natural language understanding: A survey," *arXiv preprint arXiv:2208.11857*, 2022.
[18] G. Todd, S. Earle, M. U. Nasir, M. C. Green, and J. Togelius, "Level generation through large language models," in *Proceedings of the 18th International Conference on the Foundations of Digital Games*, 2023, pp. 1–8.
[19] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.
[20] L. Wu, Z. Zheng, Z. Qiu, H. Wang, H. Gu, T. Shen, C. Qin, C. Zhu, H. Zhu, Q. Liu *et al.*, "A survey on large language models for recommendation," *arXiv preprint arXiv:2305.19860*, 2023.
[21] W. Fan, Z. Zhao, J. Li, Y. Liu, X. Mei, Y. Wang, J. Tang, and Q. Li, "Recommender systems in the era of large language models (llms)," *arXiv preprint arXiv:2307.02046*, 2023.

[22] J. Lin, X. Dai, Y. Xi, W. Liu, B. Chen, X. Li, C. Zhu, H. Guo, Y. Yu, R. Tang *et al.*, "How can recommender systems benefit from large language models: A survey," *arXiv preprint arXiv:2306.05817*, 2023.

[23] Y. Hou, J. Zhang, Z. Lin, H. Lu, R. Xie, J. McAuley, and W. X. Zhao, "Large language models are zero-shot rankers for recommender systems," *arXiv preprint arXiv:2305.08845*, 2023.

[24] J. Liu, C. Liu, R. Lv, K. Zhou, and Y. Zhang, "Is chatgpt a good recommender? a preliminary study," *arXiv preprint arXiv:2304.10149*, 2023.

[25] L. Wang and E.-P. Lim, "Zero-shot next-item recommendation using large pretrained language models," *arXiv preprint arXiv:2304.03153*, 2023.

[26] J. Zhang, R. Xie, Y. Hou, W. X. Zhao, L. Lin, and J.-R. Wen, "Recommendation as instruction following: A large language model empowered recommendation approach," *arXiv preprint arXiv:2305.07001*, 2023.

[27] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[28] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," *arXiv preprint arXiv:1301.7363*, 2013.

[29] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.

[30] W. Wei, X. Ren, J. Tang, Q. Wang, L. Su, S. Cheng, J. Wang, D. Yin, and C. Huang, "Llmrec: Large language models with graph augmentation for recommendation," *arXiv preprint arXiv:2311.00423*, 2023.

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

[32] X. Ren, W. Wei, L. Xia, L. Su, S. Cheng, J. Wang, D. Yin, and C. Huang, "Representation learning with large language models for recommendation," *arXiv preprint arXiv:2310.15950*, 2023.

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

[34] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," *arXiv preprint arXiv:1205.2618*, 2012.

[35] J. Yang, X. Yi, D. Zhiyuan Cheng, L. Hong, Y. Li, S. Xiaoming Wang, T. Xu, and E. H. Chi, "Mixed negative sampling for learning two-tower neural networks in recommendations," in *Companion Proceedings of the Web Conference 2020*, 2020, pp. 441–447.

[36] J. L. Hein, *Discrete mathematics*. Jones & Bartlett Learning, 2003.

[37] S. Zhang, L. Dong, X. Li, S. Zhang, X. Sun, S. Wang, J. Li, R. Hu, T. Zhang, F. Wu *et al.*, "Instruction tuning for large language models: A survey," *arXiv preprint arXiv:2308.10792*, 2023.

[38] K. Bao, J. Zhang, Y. Zhang, W. Wang, F. Feng, and X. He, "Tallrec: An effective and efficient tuning framework to align large language model with recommendation," *arXiv preprint arXiv:2305.00447*, 2023.

[39] S. Luo, C. Ma, Y. Xiao, and L. Song, "Improving long-tail item recommendation with graph augmentation," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023, pp. 1707–1716.

[40] Y.-J. Park and A. Tuzhilin, "The long tail of recommender systems and how to leverage it," in *Proceedings of the 2008 ACM conference on Recommender systems*, 2008, pp. 11–18.

[41] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the royal statistical society. series c (applied statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[42] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[43] C. Chen, W. Ma, M. Zhang, C. Wang, Y. Liu, and S. Ma, "Revisiting negative sampling vs. non-sampling in implicit recommendation," *ACM Transactions on Information Systems*, vol. 41, no. 1, pp. 1–25, 2023.

[44] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM conference on recommender systems*, 2016, pp. 191–198.

[45] P. Wang, L. Li, L. Chen, D. Zhu, B. Lin, Y. Cao, Q. Liu, T. Liu, and Z. Sui, "Large language models are not fair evaluators," *arXiv preprint arXiv:2305.17926*, 2023.

[46] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *arXiv preprint arXiv:2306.05685*, 2023.

[47] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[48] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford alpaca: An instruction-following llama model," https://github.com/tatsu-lab/stanford_alpaca, 2023.

[49] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.

[50] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," in *Proceedings of the 14th international conference on World Wide Web*, 2005, pp. 22–32.

[51] S. Luo, X. Zhang, Y. Xiao, and L. Song, "Hysage: A hybrid static and adaptive graph embedding network for context-drifting recommendations," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 1389–1398.

[52] T. Huang, Y. Dong, M. Ding, Z. Yang, W. Feng, X. Wang, and J. Tang, "Mixgcf: An improved training method for graph neural network-based recommender systems," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 665–674.

[53] J. Yu, H. Yin, X. Xia, T. Chen, J. Li, and Z. Huang, "Self-supervised learning for recommender systems: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[54] X. Xie, F. Sun, Z. Liu, S. Wu, J. Gao, J. Zhang, B. Ding, and B. Cui, "Contrastive learning for sequential recommendation," in *2022 IEEE 38th international conference on data engineering (ICDE)*. IEEE, 2022, pp. 1259–1273.

[55] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[56] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

[57] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," *arXiv preprint arXiv:2309.06180*, 2023.

[58] S. Dai, N. Shao, H. Zhao, W. Yu, Z. Si, C. Xu, Z. Sun, X. Zhang, and J. Xu, "Uncovering chatgpt's capabilities in recommender systems," *arXiv preprint arXiv:2305.02182*, 2023.