arXiv:2305.08845v1 [cs.IR] 15 May 2023

Large Language Models are Zero-Shot Rankers for Recommender Systems

Yupeng Hou[1†] **Junjie Zhang**[1†] **Zihan Lin**[2] **Hongyu Lu**[3] **Ruobing Xie**[3]
Julian McAuley[4] **Wayne Xin Zhao**[1✉]
[1]Gaoling School of Artificial Intelligence, Renmin University of China
[2]School of Information, Renmin University of China
[3]WeChat, Tencent [4]UC San Diego
`{houyupeng,junjie.zhang}@ruc.edu.cn` `batmanfly@gmail.com`

Abstract

Recently, large language models (LLMs) (*e.g.,* GPT-4) have demonstrated impressive general-purpose task-solving abilities, including the potential to approach recommendation tasks. Along this line of research, this work aims to investigate the capacity of LLMs that act as the ranking model for recommender systems. To conduct our empirical study, we first formalize the recommendation problem as a conditional ranking task, considering sequential interaction histories as *conditions* and the items retrieved by the candidate generation model as *candidates*. We adopt a specific prompting approach to solving the ranking task by LLMs: we carefully design the prompting template by including the sequential interaction history, the candidate items, and the ranking instruction. We conduct extensive experiments on two widely-used datasets for recommender systems and derive several key findings for the use of LLMs in recommender systems. We show that LLMs have promising zero-shot ranking abilities, even competitive to or better than conventional recommendation models on candidates retrieved by multiple candidate generators. We also demonstrate that LLMs struggle to perceive the order of historical interactions and can be affected by biases like position bias, while these issues can be alleviated via specially designed prompting and bootstrapping strategies. The code to reproduce this work is available at `https://github.com/RUCAIBox/LLMRank`.

1 Introduction

In the literature of recommender systems, most existing models are trained with user behavior data from a specific domain or task scenario [25, 13, 14], and often suffer from two major issues. Firstly, it is difficult to explicitly understand the real user preference, since existing models mainly capture user preference from historical interaction behaviors, *e.g.,* clicked item sequences [14, 19, 39, 17], limiting the expressive power to model the complicated user interests (*e.g.,* user intentions expressed in natural language). Secondly, these models are essentially "*narrow experts*", lacking more comprehensive knowledge in solving complicated recommendation tasks that rely on background or commonsense knowledge [11].

To improve recommendation performance and interactivity, there have been increasing efforts that explore the use of pre-trained language models (PLMs) in recommender systems [10, 18, 31]. They

---

† Equal contribution.
✉ Corresponding author.

Preprint. Under review.

aim to explicitly capture user preference in natural language [10] or transfer rich world knowledge from text corpora [18, 16]. Despite their effectiveness, thoroughly fine-tuning the recommendation models on task-specific data is still a necessity, making it less capable of solving diverse recommendation tasks [18]. More recently, large language models (LLMs) have shown superior capabilities in commonsense reasoning, knowledge utilization, and task generalization [37], which have shown great potential to serve as zero-shot task solvers [32, 27]. Indeed, there are some preliminary attempts that employ LLMs for solving recommendation tasks [9, 29, 30, 5, 21, 34]. These studies mainly focus on discussing the possibility of building a capable recommender with LLMs, and report promising results based on preliminary experiments. While our focus is to take a more detailed and in-depth analysis of such abilities, and understand the factors in possessing them, *e.g.,* how LLMs learn from historical interaction data.

In this paper, we aim to investigate the capacity of LLMs that serve as recommendation models by conducting a more detailed empirical study. Typically, recommender systems are developed in a pipeline architecture [4], consisting of multi-stage candidate generation (*retrieving more relevant items*) and ranking (*ranking relevant items at a higher position*) procedures. This work mainly focuses on the ranking stage of recommender systems, since LLMs are more expensive to run on a large-scale candidate set. Further, the ranking performance is sensitive to the retrieved top-ranked candidate items, which is more suitable to examine the subtle differences in the recommendation abilities of LLMs.

To carry out this study, we first formalize the recommendation process of LLMs as a *conditional ranking* task. Given prompts that include sequential historical interactions as *"conditions"*, LLMs are instructed to rank a set of *"candidates"* (*e.g.,* items retrieved by candidate generation models), according to LLM's intrinsic knowledge about the relationships between candidate items and historically interacted items. Then we conduct controlled experiments to systematically study the empirical performance of LLMs as rankers by designing specific configurations for "conditions" and "candidates", respectively. Overall, we attempt to answer the following key questions:

- Can LLMs capture underlying user preferences from prompts with *sequential* interactions?
- Can LLMs leverage their intrinsic knowledge to rank candidates retrieved by different practical strategies?

Our empirical experiments are conducted on two widely-used public datasets for recommender systems. Our experiments lead to several key findings that potentially shed light on how to develop LLMs as powerful ranking models for recommender systems. We summarize the key findings of this empirical study as follows:

- LLMs can utilize historical behaviors for personalized ranking, but *struggle to perceive the order* of the given sequential interaction histories.
- By employing specifically designed promptings, such as recency-focused prompting and in-context learning, *LLMs can be triggered to perceive the order* of sequential historical interactions, leading to improved ranking performance.
- LLMs outperform existing zero-shot recommendation methods, showing promising zero-shot ranking abilities, especially on candidates retrieved by multiple candidate generation models with different practical strategies.
- LLMs suffer from position bias and popularity bias while ranking, which can be alleviated by prompting or bootstrapping strategies.

2 General Framework for LLMs as Rankers

To investigate the recommendation abilities of LLMs, we first formalize the recommendation process as a conditional ranking task. Then, we describe a general framework that adapts LLMs to solve the recommendation task.

2.1 Problem Formulation

Given the historical interactions $\mathcal{H} = \{i_1, i_2, \ldots, i_n\}$ of one user (in chronological order of interaction time) as *conditions*, the task is to rank the *candidate* items $\mathcal{C} = \{i_j\}_{j=1}^{m}$, such that the items of

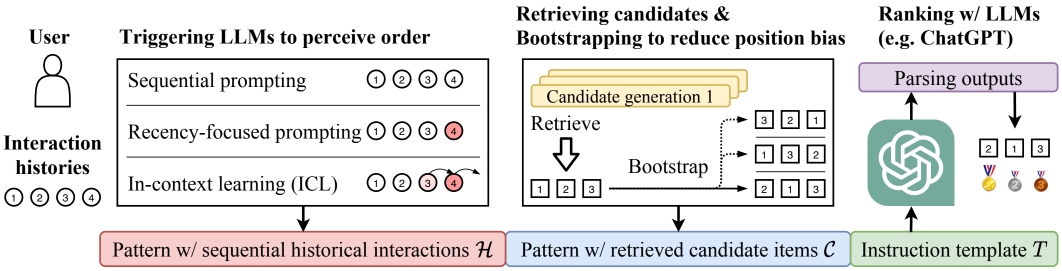

User
Interaction histories
① ② ③ ④
Triggering LLMs to perceive order
Sequential prompting ① ② ③ ④
Recency-focused prompting ① ② ③ ④
In-context learning (ICL) ① ② ③ ④
Pattern w/ sequential historical interactions $\mathcal{H}$
Retrieving candidates &
Bootstrapping to reduce position bias
Candidate generation 1
Retrieve
1 2 3
Bootstrap
3 2 1
1 3 2
2 1 3
Pattern w/ retrieved candidate items $\mathcal{C}$
Ranking w/ LLMs
(e.g. ChatGPT)
Parsing outputs
2 1 3
Instruction template T

Figure 1: An overview of the proposed LLM-based zero-shot personalized ranking method.

interest would be ranked at a higher position. In practice, the candidate items are usually retrieved by candidate generation models from the whole item set $\mathcal{I}$ ($m \ll |\mathcal{I}|$) [4]. Further, we assume that each item i is associated with a descriptive text t_i following [18].

2.2 Ranking with LLMs Using Natural Language Instructions

We use LLMs as ranking models to solve the above-mentioned task in an instruction-following paradigm [32]. Specifically, for each user, we first construct two natural language patterns that contain sequential interaction histories $\mathcal{H}$ (*conditions*) and retrieved candidate items $\mathcal{C}$ (*candidates*), respectively. Then these patterns are filled into a natural language template T as the final instruction. In this way, LLMs are expected to understand the instructions and output the ranking results as the instruction suggests. The overall framework of the ranking approach by LLMs is depicted in Figure 1. Next, we describe the detailed instruction design in our approach.

Sequential historical interactions. To investigate whether LLMs can capture user preferences from historical user behaviors, we include sequential historical interactions $\mathcal{H}$ into the instructions as inputs of LLMs. To enable LLMs to be aware of the sequential nature of historical interactions, we propose three ways to construct the instructions:

- **Sequential prompting**: Arrange the historical interactions in chronological order. This way has also been used in prior studies [5]. For example, *"I've watched the following movies in the past in order: '0. Multiplicity', '1. Jurassic Park', ..."*.
- **Recency-focused prompting**: In addition to the sequential interaction records, we can add an additional sentence to emphasize the most recent interaction. For example, *"I've watched the following movies in the past in order: '0. Multiplicity', '1. Jurassic Park', ... Note that my most recently watched movie is Dead Presidents. ..."*.
- **In-context learning (ICL)**: ICL is a prominent prompting approach for LLMs to solve various tasks [37], where it includes demonstration examples (possibly with the task description) in the prompt and instructs LLMs to solve a specific task. For the personalized recommendation task, simply introducing examples of other users may introduce noises because different users usually have different preferences. By adapting ICL in our setting, we introduce demonstration examples by augmenting the input interaction sequence itself. In detail, we pair the prefix of the input interaction sequence and the corresponding successor as examples. For example, *" If I've watched the following movies in the past in order: '0. Multiplicity', '1. Jurassic Park', ..., then you should recommend Dead Presidents to me and now that I've watched Dead Presidents, then ..."*.

Retrieved candidate items. Typically, candidate items to be ranked are first retrieved by several candidate generation models [4]. To rank these candidates with LLMs, we also arrange the candidate items $|\mathcal{C}|$ in a sequential manner. For example, *"Now there are 20 candidate movies that I can watch next: '0. Sister Act', '1. Sunset Blvd', ..."*. Note that, following the classic candidate generation approach [4], there is no specific order for candidate items. We apply a set union for the retrieved results of different candidate generation models, and randomly assign the position to a candidate item. In this work, we consider a relatively small pool for the candidates, and keep 20 candidate items (*i.e.,* $m = 20$) for ranking. It has been shown that LLMs are sensitive to the order of

Table 1: Statistics of the datasets after preprocessing. "Avg. $|\mathcal{H}|$" denotes the average length of historical user behaviors. "Avg. $|t_i|$" denotes the average number of tokens in the descriptive text of the items.

Dataset	#Users	#Items	#Interactions	Sparsity	Avg. $\lvert\mathcal{H}\rvert$	Avg. $\lvert t_i\rvert$
ML-1M	6,040	3,706	1,000,209	95.53%	46.19	16.96
Games	50,547	16,859	389,718	99.95%	7.02	43.31

examples in prompts [38, 22]. As a result, We generate different orders for the candidate items in the prompts, which enables us to further examine whether the ranking results of LLMs are affected by the arrangement order of candidates, *i.e.,* position bias, and how to alleviate position bias via bootstrapping.

Ranking with large language models. Existing studies show that LLMs can follow natural language instructions to solve diverse tasks in a zero-shot or few-shot setting [32, 37]. To use LLMs as ranking models, we finally integrate the above-mentioned patterns into the instruction template T. An example instruction template can be given as: "*[pattern that contains sequential historical interactions $\mathcal{H}$] [pattern that contains retrieved candidate items $\mathcal{C}$] Please rank these movies by measuring the possibilities that I would like to watch next most, according to my watching history. You MUST rank the given candidate movies. You cannot generate movies that are not in the given candidate list.*".

Parsing the output of LLMs. By feeding the instructions into LLMs, we can obtain the ranking results of LLMs for recommendation. Note that the output of LLMs is still in natural language text, and we parse the output with heuristic text-matching methods and ground the recommendation results on the specified item set. In detail, when the text of items is short and discriminative, like movie titles, we can directly perform efficient substring matching algorithms like KMP [20] between the LLM outputs and the text of candidate items. Otherwise, we can assign an index for each candidate item and instruct LLMs to directly output the ranked indices. Despite that candidate items are included in the prompts, we have found that LLMs have a tendency to generate items that are out of the candidate set. While, the proportion of this error is very small for GPT-3.5, about 3%. In this case, we can either remind LLMs of this error or simply treat it as an incorrect recommendation.

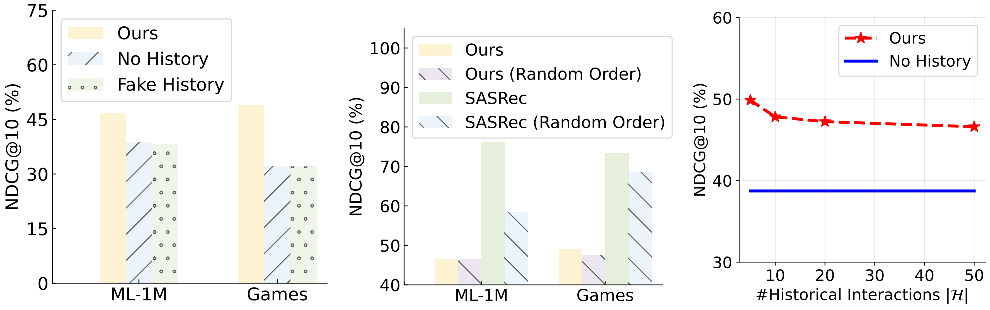
3 Empirical Studies

We aim to examine the effect of various configurations, including sequential historical interactions $\mathcal{H}$, candidates $\mathcal{C}$, and template $\mathcal{T}$, and focus on answering two research questions: (a) can LLMs capture user preferences from prompts with user sequential historical interactions $\mathcal{H}$? (b) can LLMs leverage their intrinsic knowledge to rank candidates $\mathcal{C}$ retrieved by different practical strategies?

Datasets. The experiments are conducted on two widely-used public datasets for recommender systems: (1) the movie rating dataset *MovieLens-1M* [12] (in short, **ML-1M**) where user ratings are regarded as interactions, and (2) one category from the *Amazon Review* dataset [23] named **Games** where reviews are regarded as interactions. We filter out users and items with fewer than five interactions. Then we sort the interactions of each user by timestamp, with the oldest interactions first, to construct the corresponding historical interaction sequences. The movie/product titles are used as the descriptive text of an item.

Evaluation configurations. Following existing works [19, 18, 17], we apply the leave-one-out strategy for evaluation. For each historical interaction sequence, the last item is used as the ground-truth item. We adopt the widely used metric NDCG@N to evaluate the ranking results over the given m candidates, where $N \leq m$.

Implementation details. To ease the reproduction of this work, our experiments are conducted using a popular open-source recommendation library RECBOLE [36, 35, 33]. For sequential historical user behaviors, we use the most recent 50 interactions by default. For LLM-based methods, we randomly sample 200 users along with their historical behaviors for each dataset. For conventional



(a) Use History (b) Perceive Order (c) History Length (ML-1M)

Figure 2: Analysis of whether LLMs make use of historical user behaviors and whether LLMs perceive the order of interaction histories.

baseline methods like SASRec [19], they are trained on all the interactions in the training dataset unless specified, the evaluated LLM is accessed by calling OpenAI's API `gpt-3.5-turbo`[1]. The hyperparameter temperature of calling LLMs is set to 0.2. All the reported results are the average of at least three repeat runs to reduce the effect of randomness.

3.1 Can LLMs Understand Prompts that Involve Sequential Historical User Behaviors?

In existing literature, historical user behaviors are mainly modeled as graphs [13] or sequences [14] by specially designed recommendation models. In contrast, our work encodes historical user behaviors as the prompts and feeds them into large language models not specifically trained for recommendation. In this part, we first investigate whether LLMs can leverage these historical user behaviors for making accurate recommendations. By designing different configurations of $\mathcal{H}$, we aim to examine: (1) whether LLMs can understand prompts with historical behaviors and rank correspondingly, (2) whether the sequential nature is perceived and utilized for understanding user preferences, and (3) whether LLMs can make better use of long-range user histories.

As experiments in this section mainly focus on the effect of historical user behaviors, we employ a simple strategy for constructing the candidate sets to evaluate the LLMs' ranking performance. Specifically, for each ground-truth item, we randomly retrieve $m - 1$ items from the entire item set $\mathcal{I}$ as negative instances, where $m = 20$. These candidate items are then randomly shuffled before constructing the prompts.

LLMs can give personalized recommendations corresponding to prompts with historical behaviors. In this section, we examine whether LLMs can understand prompts with historical user behaviors and give personalized recommendations. Given prompts with sequential user historical behaviors, the task is to rank a candidate set of 20 items, including one ground-truth item and 19 randomly sampled negatives. By analyzing historical behaviors, items of interest should be ranked at a higher position. We compare the ranking results of three LLM-based methods: (a) *Ours*, which ranks with LLMs as we have described in Section 2.2. Historical user behaviors are encoded into prompts using the "sequential prompting" strategy. (b) *No History*, where the historical user behaviors are removed from instructions, and (c) *Fake History*, where we replace all the items in original historical behaviors with randomly sampled items as fake historical behaviors.

From Figure 2(a), we can see that *Ours* has better performance than variants with no historical behaviors or fake historical behaviors. The results suggest that LLMs can effectively leverage prompts with historical user behaviors to make personalized recommendations, while unrelated historical behaviors may also hurt the ranking performance of LLMs.

LLMs struggle to perceive the order of the given historical user behaviors. In Figure 2(b), we further investigate the ability of LLMs to recognize the sequential nature of user historical behaviors. The variant with the suffix *(Random Order)* refers to shuffling the historical user behaviors randomly

---

[1] `https://openai.com/blog/introducing-chatgpt-and-whisper-apis`

Table 2: Performance comparison of different zero-shot recommendation models on *randomly retrieved candidates*. Ground-truth items are guaranteed to be included in the candidate sets. "full" denotes recommendation models that are trained on the target dataset, and "zero-shot" denotes recommendation models that are not trained on the target dataset but could be pre-trained. The three zero-shot prompting strategies are based on `gpt-3.5-turbo`. We highlight the best performance among zero-shot recommendation methods in **bold**. N@K denotes NDCG@K.

	Method	ML-1M				Games			
		N@1	N@5	N@10	N@20	N@1	N@5	N@10	N@20
full	Pop	19.50	40.70	48.16	51.78	25.50	45.16	50.75	55.58
	BPRMF [25]	32.50	54.07	60.17	61.93	39.50	59.07	62.66	65.92
	SASRec [19]	57.00	73.14	76.25	77.47	52.50	70.71	73.25	74.74
zero-shot	BM25 [26]	4.00	13.14	20.53	33.70	9.00	22.85	31.08	40.55
	UniSRec [18]	9.00	20.08	26.72	38.24	22.50	37.74	42.64	51.03
	VQ-Rec [16]	9.50	19.52	27.11	38.72	8.00	19.36	29.43	39.06
	Sequential	21.50	40.71	46.61	52.24	23.17	44.10	49.06	53.62
	Recency-Focused	**23.33**	42.07	48.80	53.73	**23.83**	**45.69**	**50.31**	**55.45**
	In-Context Learning	22.67	**44.51**	**49.97**	**54.60**	19.67	45.30	49.68	54.05

before feeding to the model (either *Ours* or *SASRec*). By comparing *SASRec* and *SASRec (Random Order)*, we can see that the order of sequential historical interactions is vital for the item ranking. However, the performance of *Ours* and *Ours (Random Order)* is quite similar, indicating that LLMs are not sensitive to the order of given historical user behaviors.

Moreover, in Figure 2(c), we vary the number of latest historical user behaviors ($|\mathcal{H}|$) used for constructing the prompt from 5 to 50. The results show that increasing the number of historical user behaviors does not improve the ranking performance, but even negatively impacts the ranking performance. We speculate that this phenomenon is caused by the fact that LLMs have difficulty understanding the order, but consider all the historical behaviors equally. Therefore too many historical user behaviors (*e.g.,* $|\mathcal{H}| = 50$) may overwhelm LLMs and lead to a performance drop. In contrast, a relatively small $|\mathcal{H}|$ enables LLMs to concentrate on the most recently interacted items, resulting in better recommendation performance. The above results can be summarized as the first key observation:

Observation 1. LLMs can utilize historical behaviors for personalized ranking, but *struggle to perceive the order* of the given sequential interaction histories.

Triggering LLMs to perceive the interaction order. Based on the above observations, we find that it is difficult for LLMs to perceive the order in interaction histories by a sequential prompting strategy. As a result, we propose two alternative prompting strategies, aiming to elicit the order-perceiving abilities of LLMs. The core idea is to emphasize the recently interacted items. Detailed descriptions of the proposed recency-focused prompting and in-context learning strategies have been given in Section 2.2.

In Table 2, we can see that both recency-forced prompting and in-context learning can improve the ranking performance of LLMs. Recency-focused prompting yields better top-1 accuracies, while in-context learning performs better on datasets with longer historical behaviors. The above results can be summarized as the following key observation:

Observation 2. By employing specifically designed promptings, such as recency-focused prompting and in-context learning, *LLMs can be triggered to perceive the order* of historical user behaviors, leading to improved ranking performance.

3.2 How Well Can LLMs Rank Candidate Items in a Zero-Shot Setting?

In this section, we further investigate how well can LLMs rank the candidates. We first conduct benchmarking experiments to compare the ranking performance between different methods on random candidates, including conventional recommendation models, existing zero-shot recommendation

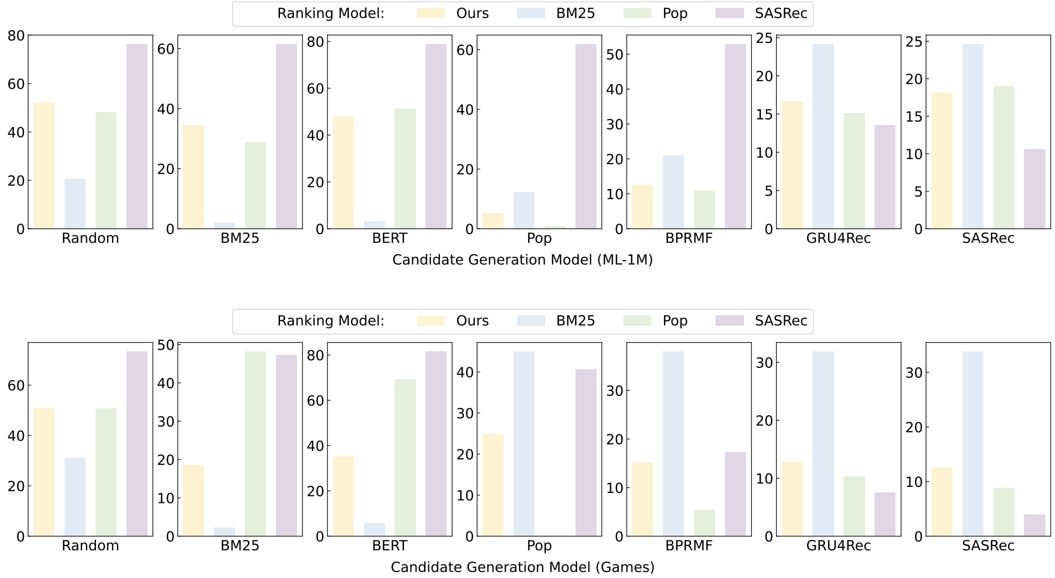


Figure 3: Ranking performance measured by NDCG@10 (%) on hard negatives retrieved by different strategies.

methods, and the proposed LLM-based methods. Next, we evaluate LLM-based methods on candidates with hard negatives that are retrieved by different strategies to further investigate what does the ranking of LLMs depend on? Then, we present another benchmark to compare the ranking performance of different methods on candidates retrieved by multiple candidate generation models to simulate a more practical and difficult setting.

LLMs have promising zero-shot ranking abilities. In Table 2, we conduct experiments to compare the ranking abilities of LLM-based methods with existing methods. We follow the same setting in Section 3.1 where $|\mathcal{C}| = 20$ and candidate items (other than the ground-truth item) are randomly retrieved. We include three conventional recommendation models that are trained on the training set, *i.e.,* Pop (recommending according to item popularity), BPRMF [25], and SASRec [19]. We also evaluate three zero-shot recommendation methods that are not trained on the target datasets, including BM25 [26] (rank according to the textual similarity between candidates and historical interactions), UniSRec [18], and VQ-Rec [16]. For UniSRec and VQ-Rec, we use their publicly available pre-trained models. We do not include ZESRec [7] because there is no pre-trained model released, and UniSRec can be regarded as an improved version of this model [18]. For LLM-based methods, we include the three variants that use different prompting strategies as described in Section 2.2, named Sequential.

From Table 2, we can see that LLM-based methods outperform existing zero-shot recommendation methods by a large margin, showing promising zero-shot ranking abilities. We would highlight that it is difficult to conduct zero-shot recommendations on the ML-1M dataset, due to the difficulty in measuring the similarity between movies merely by the similarity of their titles. We can observe that LLM-based models still achieve promising zero-shot ranking performance on ML-1M, as they can use intrinsic world knowledge to measure the similarity between movies and make recommendations. However, we can also see that there are still gaps between zero-shot recommendation methods and conventional methods, indicating the importance to develop LLM-based recommendation methods that can learn from interaction data [34].

LLMs rank candidates based on item popularity, text features as well as user behaviors. To further investigate how LLMs rank the given candidates, we evaluate the ranking performance of LLMs on candidates that are retrieved by different candidate generation methods. These candidates can be viewed as hard negatives for ground-truth items, which can be used to measure the ranking ability of LLMs for specific categories of items. We consider two categories of strategies to retrieve

Table 3: Performance comparison of different zero-shot recommendation models on *candidates retrieved by multiple candidate generation models*. Ground-truth items are *not* guaranteed to be included in the candidate sets. “full” denotes recommendation models that are trained on the target dataset, and “zero-shot” denotes recommendation models that are not trained on the target dataset but could be pre-trained. We highlight the best performance among *all* recommendation methods in **bold**. N@K denotes NDCG@K.

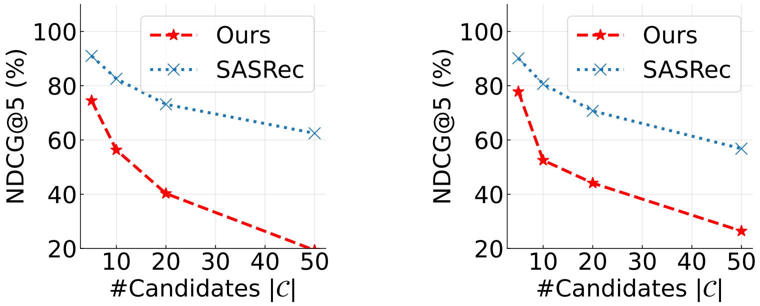
	Method	ML-1M				Games			
		N@1	N@5	N@10	N@20	N@1	N@5	N@10	N@20
full	Pop	0.00	1.19	3.32	5.28	0.00	0.39	1.57	1.69
	BPRMF [25]	1.50	3.01	4.78	6.39	0.50	0.72	1.03	1.82
	SASRec [19]	2.00	**7.13**	**8.38**	8.52	0.00	2.23	2.56	2.56
zero-shot	BM25 [26]	0.50	0.75	2.20	4.95	0.00	0.60	1.10	1.63
	UniSRec [18]	1.50	4.09	5.37	6.77	0.00	1.86	2.03	2.31
	VQ-Rec [16]	0.00	1.68	3.52	5.22	0.00	0.76	1.12	1.73
	Ours	**3.83**	6.20	7.81	**8.65**	**1.00**	**2.24**	**2.57**	**2.74**

the candidates: (1) *content-based methods* like *BM25* [26] and *BERT* [6] retrieve candidates based on the text feature similarities between historical interacted items and candidates, and (2) *interaction-based methods*, including *Pop* (recommend based on item popularity), *BPRMF* [25], *GRU4Rec* [14], and *SASRec* [19], retrieve items using conventional recommendation models trained on user-item interactions. Given candidates, we compare the ranking performance of the LLM-based model (*Ours*) and representative content-based (*BM25*) and interaction-based (*Pop* and *SASRec*) methods.

From Figure 3 we can see that the ranking performance of the LLM-based method varies on different candidate sets and different datasets. (1) On ML-1M, our LLM-based method cannot rank well on candidate sets that contain popular items (*e.g., Pop* and *BPRMF*), indicating the LLM-based method recommend items largely depend on item popularity on ML-1M dataset. (2) On Games, we can observe that *Ours* has similar ranking performance both on popular candidates and textual similar candidates, showing that item popularity and text features contribute similarly to the ranking of LLMs. (3) On both two datasets, the ranking performance of *Ours* is affected by hard negatives retrieved by interaction-based candidate generation methods, but not as severe as those ranking models that are purely based on interactions like *SASRec*. The above results demonstrate that LLM-based methods not only consider some single aspect for ranking, but make use of item popularity, text features, and even user behaviors. On different datasets, the weights of these three aspects to affect the ranking performance may also vary.

LLMs can effectively rank candidates retrieved by multiple candidate generation models. For real-world two-stage recommender systems [4], the items to be ranked are usually retrieved by multiple candidate generation models. As a result, we also conduct benchmarking experiments in a more practical setting. We use seven candidate generation models to retrieve items, *i.e., Random*, *BM25*, *BERT*, *Pop*, *BPRMF*, *GRU4Rec*, and *SASRec*, covering typical content-based and interaction-based methods. The top-3 best items retrieved by each candidate generation model will be merged into a candidate set containing a total of 21 items. Note that as a more practical setting, we do not complement the ground-truth item to each candidate set like the setting described in Section 3.1. For *Ours*, inspired by experiments in Section 3.1, we use the recency-focused prompting strategy to encode $|\mathcal{H}| = 5$ sequential historical interactions into prompts for a decent ranking performance.

From Table 3, we can see that the LLM-based ranking model (*Ours*) yields the best performance over the compared recommendation models on most metrics (6 of 8), even beats the conventional recommendation model *SASRec* that has been trained on the target datasets. The results demonstrate the strong zero-shot ranking ability of LLMs on candidates retrieved by multiple candidate generation models. Facing the phenomenon, we assume that LLMs can make use of their intrinsic world knowledge to rank the candidates comprehensively considering popularity, text features, and user behaviors. In comparison, existing models (as *narrrow experts*) may lack the ability to rank items in a complicated setting. The above findings can be summarized as:



(a) ML-1M (b) Games

Figure 4: Ranking performance comparison between LLM-based model (*Ours*) and conventional recommendation model (*SASRec*) on different sizes of candidate sets.

Observation 3. LLMs have promising zero-shot ranking abilities, especially on candidates retrieved by multiple candidate generation models with different practical strategies.

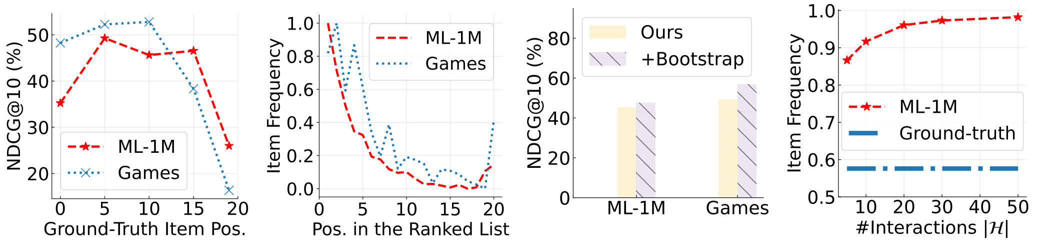
LLMs cannot rank candidates well when the candidate set is large. It has been a technical challenge to effectively model the semantics of long sequences by language models [8]. As a result, we would like to investigate whether LLMs can deal with a large set of candidates for ranking. We vary the number of candidates $|\mathcal{C}|$ from 5 to 50 and report the ranking performance in Figure 4. We can see that the gap between LLMs and conventional recommendation models (*e.g.,* SASRec) enlarges as $|\mathcal{C}|$ increases, indicating that LLMs may face challenges when ranking a large set of candidate items.

3.3 Do LLMs suffer from biases while ranking?

The biases and debiasing methods in conventional recommender systems have been widely studied [2]. For the proposed LLM-based zero-shot recommendation model, there could also be specific biases that affect the ranking of LLMs. In this section, we discuss two kinds of biases that LLM-based recommendation models suffer from, namely position bias and popularity bias. We also make discussions on how to alleviate these biases.

The order of candidates affects the ranking results of LLMs. For conventional ranking methods, the order of retrieved candidates usually will not affect the ranking results. However, for our LLM-based approach that is described in Section 2.2, the candidate items are arranged in a sequential manner and encoded into a prompt as inputs of LLMs. It has been shown that LLMs are generally sensitive to the order of examples in the prompts for NLP tasks [38, 22]. As a result, we also conduct experiments to examine whether the order of candidates affects the ranking performance of LLMs. We evaluate the performance of LLMs on the same candidate sets that are used in Section 3.1. The only difference is that we control the order of these candidates in the prompts by purpose, *i.e.,* we make the ground-truth items appear at a certain position while constructing prompts. We vary the position of ground-truth items at $\{0, 5, 10, 15, 19\}$ and present the ranking results in Figure 5(a). We can see that the ranking performance varies when the ground-truth items appear at different positions. Specifically, the ranking performance drops significantly when the ground-truth items appear at the last few positions. The results indicate that the ranking performance of LLMs is affected by the order of candidates, *i.e., position bias* for LLM-based rankers, while conventional recommendation models are usually *not* influenced.

Alleviating position bias via bootstrapping. From Figure 5(a), we can see that LLMs tend to rank the candidate items lower if they locate at a later position in the prompts. As the candidate items are randomly assigned to each position, a simple strategy to alleviate position bias is to bootstrap the ranking process. We may rank the candidate set repeatedly for B times, with candidates randomly shuffled at each round, so that one candidate item may appear at different positions to be ranked.



(a) Position Bias (b) Popularity Bias (c) Bootstrapping (d) $|\mathcal{H}|$ *v.s.* popularity of the best ranked item

Figure 5: Biases and debiasing methods in the ranking of LLMs. (a) The position of candidates in the prompts influences the ranking results. (b) LLMs tend to recommend popular items. (c) Bootstrapping alleviates position bias. (d) Focusing on historical interactions reduces popularity bias.

After ranking, the item at a higher position will be given a higher score, and we will merge the ranking scores together to derive the final ranking. From Figure 5(c), we follow the setting in Section 3.1 and apply the bootstrapping strategy to *Ours*. Each candidate set will be ranked for 3 times. We can see that bootstrapping improves the ranking performance on both datasets.

Popularity degrees of candidates affect ranking results of LLMs. For popular items, the associated text may also appear frequently in the pre-training corpora of LLMs. For example, a best-selling book would be widely discussed on the Web. Thus, we would like to examine whether the ranking results are affected by the popularity degrees of candidates. However, it is difficult to directly measure the item text popularity in the pre-training corpora. As a result, we hypothesize that the text popularity can be reflected and indirectly measured by item frequency in one recommendation dataset. In Figure 5(b), we report the item popularity score (measured by the normalized item frequency of appearance in the training set) at each position of the ranked item lists. We can see that popular items tend to be ranked at higher positions. Like conventional recommendation models, LLMs also suffer from popularity bias and favor recommending more popular items.

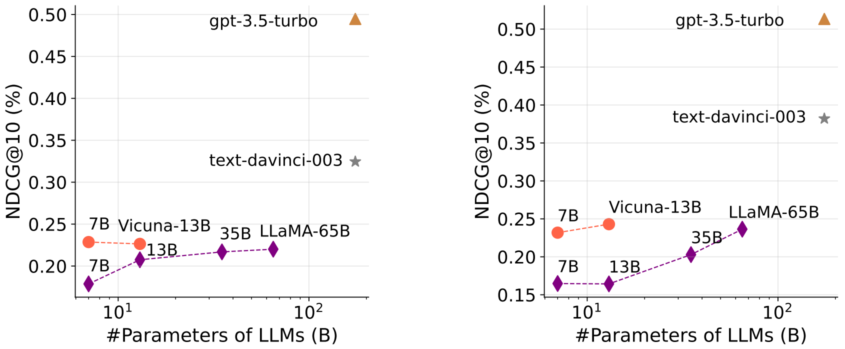
Making LLMs focus on historical interactions helps reduce popularity bias. From Figure 5(b), we can see that LLMs tend to rank popular items at higher positions. As observations in Section 3.2 indicate, the reason could be that LLMs do not leverage historical interactions well, and have to make recommendations mainly based on item popularity. From experiments in Figure 2(c), we know that LLMs can make better use of historical interactions when the number of used historical interactions is smaller. As a result, we vary the number of historical interactions to see whether popularity bias can be reduced once LLMs focus more on user histories. From Figure 5(d), we compare the popularity scores (measured by normalized item frequency) of the best-ranked items. It can be observed that as the number of historical interactions decreases, the popularity score decreases as well. This suggests that one can reduce the effects of popularity bias when LLMs are forced to focus on historical interactions. From the above experiments, we can conclude the following:

Observation 4. LLMs suffer from position bias and popularity bias while ranking, which can be alleviated by specially designed prompting or bootstrapping strategies.

3.4 How Do LLMs Gain Recommendation Abilities?

In this section, we explore what factors or techniques contribute to the ranking abilities of LLMs. Specially, we mainly consider examining the effect of two factors, namely instruction tuning and model scaling, on the ranking abilities of LLMs, because both techniques have been shown to be key to the abilities of LLMs [37, 1, 32]. We would like to take a more focused study on how they improve the recommendation performance of LLMs.

Instruction tuning improves the ranking abilities of LLMs. Existing works show that instruction tuning significantly improves the generalization abilities of LLMs on unseen tasks [27, 32]. Here we would like to investigate whether the ability to rank items according to historical interactions using



(a) ML-1M (b) Games

Figure 6: Ranking performance comparison using different LLMs.

LLMs can be improved by instruction tuning. Following experimental settings in Section 3.1, we replace the default LLM in the proposed LLM-based ranking method from `gpt-3.5-turbo` to (1) LLMs that have not been instruction-tuned, like LLaMA [28], and (2) LLMs that have been fine-tuned on instructions, including Vicuna [3] and `text-davinci-003`[2], and then instruct these LLMs to perform the ranking task. In Figure 6, by comparing Vicuna-7/13B to LLaMA-7/13B, we can see that instruction-tuned LLMs outperform LLMs that have not been instruction-tuned. The results demonstrate that instruction tuning improves the ranking abilities of LLMs, even if the instructions are not specially designed for the recommendation tasks.

Model Scaling improves the ranking performance of LLMs. As the scaling law shows, the performance of LLMs on various downstream tasks generally increases while scaling up the model size and the amount of training data [24, 15, 28]. We follow the experimental setting in Section 3.1 and replace the base LLM with LLaMA of different sizes (7B, 13B, 35B, and 65B) to investigate the effect of model scaling on the zero-shot recommendation task. From Figure 6, we can see that the ranking performance of LLMs increases as the model size increases (*i.e.,* LLaMA-65B $>$ LLaMA-35B $>$ LLaMA-13B). We can also see that LLMs larger than 100B yield superior ranking abilities, by comparing `text-davinci-003` and `gpt-3.5-turbo` with other smaller LLMs. The above results indicate that the zero-shot recommendation task also fulfills the scaling law, *i.e.,* model scaling improves the ranking performance of LLMs.

4 Conclusion

In this work, we investigated the capacities of LLMs that act as the ranking model for recommender systems. In detail, we formalized the recommendation task as a conditional ranking task, considering sequential historical interactions as conditions and the items retrieved by candidate generation models as candidates. To rank with LLMs, we further constructed natural language prompts that contain historical interactions as well as candidates. We then propose several specially designed prompting strategies to trigger the ability of LLMs to perceive orders of sequential behaviors. We also introduce a simple bootstrapping strategy to alleviate the position bias issue that LLM-based ranking models may suffer. Extensive empirical studies indicate that LLMs have promising zero-shot ranking abilities. We also conclude several key findings and aim at shedding light on several promising directions to further improve the ranking abilities of LLMs, including (1) better perceiving the order of sequential historical interactions, (2) making better use of more historical interactions and candidates, and (3) alleviating the position bias and popularity bias. For future work, we consider developing technical approaches to solve the above-mentioned key challenges when deploying LLMs as zero-shot rankers. We also would like to develop LLM-based recommendation models that can be efficiently tuned on downstream user behaviors for effective personalized recommendations.

---

[2] `https://platform.openai.com/docs/api-reference`

References

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

[2] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. Bias and debias in recommender system: A survey and future directions. *CoRR*, abs/2010.03240, 2020.

[3] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[4] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *RecSys*, pages 191–198, 2016.

[5] Sunhao Dai, Ninglu Shao, Haiyuan Zhao, Weijie Yu, Zihua Si, Chen Xu, Zhongxiang Sun, Xiao Zhang, and Jun Xu. Uncovering chatgpt's capabilities in recommender systems. *arXiv preprint arXiv:2305.02182*, 2023.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

[7] Hao Ding, Yifei Ma, Anoop Deoras, Yuyang Wang, and Hao Wang. Zero-shot recommender systems. *arXiv:2105.08318*, 2021.

[8] Zican Dong, Tianyi Tang, Lunyi Li, and Wayne Xin Zhao. A survey on long text modeling with transformers. *arXiv preprint arXiv:2302.14502*, 2023.

[9] Yunfan Gao, Tao Sheng, Youlin Xiang, Yun Xiong, Haofen Wang, and Jiawei Zhang. Chat-rec: Towards interactive and explainable llms-augmented recommender system. *arXiv preprint arXiv:2303.14524*, 2023.

[10] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. Recommendation as language processing (RLP): A unified pretrain, personalized prompt & predict paradigm (P5). In *RecSys*, 2022.

[11] Qingyu Guo, Fuzhen Zhuang, Chuan Qin, Hengshu Zhu, Xing Xie, Hui Xiong, and Qing He. A survey on knowledge graph-based recommender systems. *TKDE*, 34(8):3549–3568, 2020.

[12] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *TIIS*, 5(4):1–19, 2015.

[13] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*, 2020.

[14] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *ICLR*, 2016.

[15] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[16] Yupeng Hou, Zhankui He, Julian McAuley, and Wayne Xin Zhao. Learning vector-quantized item representation for transferable sequential recommenders. In *WWW*, 2023.

[17] Yupeng Hou, Binbin Hu, Zhiqiang Zhang, and Wayne Xin Zhao. Core: Simple and effective session-based recommendation within consistent representation space. In *SIGIR*, 2022.

[18] Yupeng Hou, Shanlei Mu, Wayne Xin Zhao, Yaliang Li, Bolin Ding, and Ji-Rong Wen. Towards universal sequence representation learning for recommender systems. In *KDD*, 2022.

[19] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *ICDM*, 2018.

[20] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[21] Guo Lin and Yongfeng Zhang. Sparks of artificial general recommender (agr): Early experiments with chatgpt. *arXiv preprint arXiv:2305.04518*, 2023.

[22] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *ACL*, 2022.

[23] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP*, pages 188–197, 2019.

[24] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[25] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. In *UAI*, 2009.

[26] Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009.

[27] Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. Multitask prompted training enables zero-shot task generalization. In *ICLR*, 2022.

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

[29] Lei Wang and Ee-Peng Lim. Zero-shot next-item recommendation using large pretrained language models. *arXiv preprint arXiv:2304.03153*, 2023.

[30] Wenjie Wang, Xinyu Lin, Fuli Feng, Xiangnan He, and Tat-Seng Chua. Generative recommendation: Towards next-generation recommender paradigm. *arXiv preprint arXiv:2304.03516*, 2023.

[31] Xiaolei Wang, Kun Zhou, Ji-Rong Wen, and Wayne Xin Zhao. Towards unified conversational recommender systems via knowledge-enhanced prompt learning. In *KDD*, 2022.

[32] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *ICLR*, 2022.

[33] Lanling Xu, Zhen Tian, Gaowei Zhang, Junjie Zhang, Lei Wang, Bowen Zheng, Yifan Li, Jiakai Tang, Zeyu Zhang, Yupeng Hou, Xingyu Pan, Wayne Xin Zhao, Xu Chen, and Ji-Rong Wen. Towards a more user-friendly and easy-to-use benchmark library for recommender systems. In *SIGIR*, 2023.

[34] Junjie Zhang, Ruobing Xie, Yupeng Hou, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. Recommendation as instruction following: A large language model empowered recommendation approach. *arXiv preprint arXiv:2305.07001*, 2023.

[35] Wayne Xin Zhao, Yupeng Hou, Xingyu Pan, Chen Yang, Zeyu Zhang, Zihan Lin, Jingsen Zhang, Shuqing Bian, Jiakai Tang, Wenqi Sun, Yushuo Chen, Lanling Xu, Gaowei Zhang, Zhen Tian, Changxin Tian, Shanlei Mu, Xinyan Fan, Xu Chen, and Ji-Rong Wen. Recbole 2.0: Towards a more up-to-date recommendation library. In *CIKM*, 2022.

[36] Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Yushuo Chen, Xingyu Pan, Kaiyuan Li, Yujie Lu, Hui Wang, Changxin Tian, Yingqian Min, Zhichao Feng, Xinyan Fan, Xu Chen, Pengfei Wang, Wendi Ji, Yaliang Li, Xiaoling Wang, and Ji-Rong Wen. Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms. In *CIKM*, 2021.

[37] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[38] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *ICML*, 2021.

[39] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *CIKM*, 2020.

A Examples on ML-1M Dataset

A.1 Sequential Prompting

- **Input:** *I’ve watched the following movies in the past in order:\n [’0. The Matrix’, ’1. Terminator 2: Judgment Day’, ’2. Roger & Me’, ’3. Fargo’, ’4. The Blair Witch Project’]\n\n Now there are 20 candidate movies that I can watch next:\n [’0. Nixon’, ’1. X-Men’, ’2. Best Men’, “3. Carlito’s Way”, ’4. Star Trek IV: The Voyage Home’, ’5. The Five Senses’, ’6. Mephisto’, ’7. Meatballs 4’, ’8. The Stupids’, ’9. Thunderball’, ’10. Body Parts’, ’11. Grumpier Old Men’, ’12. The Wedding Singer’, ’13. Ronin’, ’14. Apple, The (Sib)’, ’15. Tampopo’, ’16. The Goodbye Girl’, ’17. Force of Evil’, ’18. The Edge’, ’19. Klute’]\n Please rank these 20 movies by measuring the possibilities that I would like to watch next most, according to my watching history. Please think step by step.\n Please show me your ranking results with order numbers. Split your output with line break. You MUST rank the given candidate movies. You can not generate movies that are not in the given candidate list.*

- **Output:** *1. X-Men \n 2. Carlito’s Way \n 3. Ronin \n 4. The Edge \n 5. Nixon \n 6. Grumpier Old Men \n 7. The Wedding Singer \n 8. Thunderball \n 9. Body Parts \n 10. Star Trek IV: The Voyage Home \n 11. Tampopo \n 12. Klute \n 13. The Goodbye Girl \n 14. Force of Evil \n 15. Best Men \n 16. The Five Senses \n 17. Mephisto \n 18. Meatballs 4 \n 19. The Stupids \n 20. Apple, The (Sib)*

- **Result:** Ground-Truth *“X-Men”* **rank 0**

A.2 Recency-focused Prompting

- **Input:** *I’ve watched the following movies in the past in order:\n [’0. Eyes Wide Shut’, ’1. Fight Club’, “2. The General’s Daughter”, ’3. Gone in 60 Seconds’, ’4. Gladiator’]\n\n Now there are 20 candidate movies that I can watch next:\n[’0. For Whom the Bell Tolls’, ’1. Encino Man’, ’2. The Pope of Greenwich Village’, ’3. I Married A Strange Person’, ’4. Daylight’, ’5. Rain’, ’6. Children of a Lesser God’, “7. She’s the One”, ’8. For the Moment’, ’9. Once Upon a Time in America’, ’10. The Truth About Cats & Dogs’, “11. A Hard Day’s Night”, ’12. Jakob the Liar’, ’13. Angel Heart’, ’14. Galaxy Quest’, ’15. The Abominable Snowman’, ’16. Bait’, ’17. Warriors of Virtue’, ’18. Three to Tango’, ’19. Heaven & Earth’] \n Please rank these 20 movies by measuring the possibilities that I would like to watch next most, according to my watching history. Please think step by step.\n Note that my most recently watched movie is Gladiator. Please show me your ranking results with order numbers. Split your output with line break. You MUST rank the given candidate movies. You can not generate movies that are not in the given candidate list.*

- **Output:** *1. Once Upon a Time in America \n 2. Angel Heart \n 3. The Truth About Cats & Dogs \n 4. Galaxy Quest \n 5. Jakob the Liar \n 6. A Hard Day’s Night \n 7. The Pope of Greenwich Village \n 8. Rain \n 9. For Whom the Bell Tolls \n 10. Three to Tango \n 11. She’s the One \n 12. Bait \n 13. I Married A Strange Person \n 14. Children of a Lesser God \n 15. Warriors of Virtue \n 16. Heaven & Earth \n 17. Encino Man \n 18. Daylight \n 19. The Abominable Snowman \n 20. For the Moment*

- **Result:** Ground-Truth *“Jakob the Liar”* **rank 4**

A.3 In-context Learning

- **Input:** *I’ve watched the following movies in the past in order:\n [’0. Mirror, The (Zerkalo)’, ’1. The 39 Steps’, ’2. Sanjuro’, ’3. Trouble in Paradise’]\n\n Then if I ask you to recommend a new movie to me according to my watching history, you should recommend Shampoo and now that I’ve just watched Shampoo, there are 20 candidate movies that I can watch next:\n [’0. Manon of the Spring (Manon des sources)’, ’1. Air Bud’, ’2. Citizen Kane’, ’3. Grand Hotel’, ’4. A Very Brady Sequel’, ’5. 42 Up’, ’6. Message to Love: The Isle of Wight Festival’, ’7. Screamers’, ’8. The Believers’, ’9. Hamlet’, ’10. Cliffhanger’, ’11. Three Wishes’, ’12. Nekromantik’, ’13. Dangerous Minds’, ’14. The Prophecy’, ’15. Howling II: Your Sister Is a Werewolf’, ’16. World of Apu, The (Apur Sansar)’, ’17. The Breakfast Club’, ’18. Hoop Dreams’, ’19. Eddie’]\n Please rank these 20 movies by measuring the possibilities that I would like to watch next most, according to my watching history. Please think step by step.\n Please show me your ranking results with order numbers. Split* *your output with line break. You MUST rank the given candidate movies. You can not generate movies that are not in the given candidate list.*

- **Output:** *1. Manon of the Spring (Manon des sources) \n 2. Citizen Kane \n 3. Grand Hotel \n 4. Hamlet \n 5. The Breakfast Club \n 6. Hoop Dreams \n 7. Eddie \n 8. Three Wishes \n 9. The Prophecy \n 10. Dangerous Minds \n 11. World of Apu, The (Apur Sansar) \n 12. Cliffhanger \n 13. A Very Brady Sequel \n 14. Screamers \n 15. Howling II: Your Sister Is a Werewolf \n 16. Nekromantik \n 17. 42 Up \n 18. Message to Love: The Isle of Wight Festival \n 19. Air Bud \n 20. The Believers*

- **Result:** Ground-Truth *"World of Apu, The (Apur Sansar)"* **rank 10**

B Examples on Games Dataset

B.1 Sequential Prompting

- **Input:** *I've purchased the following products in the past in order:\n ['0. Sony PlayStation 3 Blu-ray Disc Remote', '1. PlayStation 3 160GB System with Ratchet &amp; Clank Future: A Crack in Time and SingStar Dance Party Pack - Family Bundle', '2. PS3 320GB Uncharted 3 Bundle', '3. FIFA Soccer 10 - Playstation 3']\n\n Now there are 20 candidate products that I can consider to purchase next:\n ['0. Wii 9 in 1 Sports Kit', '1. Donkey Kong', '2. Midnight Club', '3. Midnight Club', '4. SRS: Street Racing Syndicate - PlayStation 2', '5. Ultimate Action Triple Pack - PlayStation 3', '6. Thrustmaster TMX Force Feedback racing wheel for Xbox One and WINDOWS', '7. Pikmin 2 (Nintendo Selects) - Nintendo Wii', '8. Rock Candy Blueberry Boom', '9. Finding Nemo - Gamecube', '10. SteelSeries Siberia 200 Gaming Headset - Alchemy Gold (formerly Siberia v2)', '11. Konnet Xbox 360 / Slim Power Pyramid RCS series Charger, Charging and Storage Dock for FOUR Game Controllers - Black', '12. Megadream Xbox One Wireless Keyboard, Mini 2.4Ghz Qwerty Text Message Chatpad Keypad for Microsoft Xbox One and Xbox One Slim Controller - Not Compatible with Xbox One Elite and X Controller', '13. Theme Hospital - PlayStation', '14. Spongebob Hero Pants The Game 2015 - Xbox 360', '15. Megaman &amp; Bass', '16. BW&reg; 5 Pair/10 Pcs Replacement Silicone Analog Controller Joystick Thumb Stick Grips Cap Cover For PS3 / PS4 / Xbox 360 / Xbox One / Wii Game Controllers (green)', '17. Skylanders Giants Battlepack #1 - Chop Chop - Dragonfire Cannon - Shroomboom', '18. Rune Factory: Tides of Destiny - Playstation 3', '19. Donkey Kong Land III']\n Please rank these 20 products by measuring the possibilities that I would like to purchase next most, according to the given purchasing records. Please think step by step.\n Please only output the order numbers after ranking. Split these order numbers with line break.*

- **Output:** *1 \n 5 \n 6 \n 10 \n 11 \n 12 \n 16 \n 17 \n 18 \n 2 \n 0 \n 7 \n 8 \n 9 \n 13 \n 14 \n 15 \n 3 \n 4 \n 19*

- **Result:** Ground-Truth *"Midnight Club"* **rank 9**

B.2 Recency-focused Prompting

- **Input:** *I've purchased the following products in the past in order:\n ['0. PowerA DualShock 4 Charging Station for PlayStation 4', '1. Far Cry 4 - PlayStation 4', '2. WWE 2K17 - PlayStation 4', '3. Dragon Quest Builders - PlayStation 4']\n\n Now there are 20 candidate products that I can consider to purchase next:\n ['0. Ready to Rumble Boxing: Round 2', '1. Insten USB Charger Charging Cable Replacement For Nintendo NEW 3DS XL / NEW 2DS XL / 3DS XL / 3DS / 2DS / DSi / DSi XL / DSi LL', "2. Cabela's African Adventures - Xbox 360", '3. Blood Bowl - Xbox 360', '4. Skullcandy SLYR Gaming Headset, White (SMSLFY-205)', '5. HORI 3DS Protector and Pouch Set (Mario Kart 7 version)', '6. Grand Theft Auto V - PlayStation 4', "7. Yoshi's New Island Full Game Download Code - Nintendo 3DS eShop", '8. Gauntlet', '9. Solitaire &amp; Mahjong - Nintendo Wii', '10. Happy Feet 2 - Playstation 3', '11. Little Italy - Hidden Object Games [Download]', '12. Jillian Michaels Fitness Adventure - Xbox 360', '13. Dream Day Wedding: Viva Las Vegas - PC', '14. NBA Inside Drive 2003', '15. The Sims 2 - Sony PSP', '16. Xenogears - PlayStation', '17. Combat', '18. Crash Bandicoot: The Wrath of Cortex - Gamecube', '19. Sonic Mega Collection']\n Please rank these 20 products by measuring the possibilities that I would like to purchase next most, according to the given purchasing records. Please think step by step.\n Note that my most recently purchased product is Dragon Quest Builders - PlayStation 4. Please only output the order numbers after ranking. Split these order numbers with line break.*

- **Output:** *6 \n 0 \n 4 \n 2 \n 7 \n 18 \n 16 \n 3 \n 1 \n 8 \n 19 \n 13 \n 12 \n 11 \n 10 \n 5 \n 9 \n 15 \n 14 \n 17*
- **Result:** Ground-Truth *“Grand Theft Auto V - PlayStation 4”* **rank 0**

B.3 In-context Learning

- **Input:** *I’ve purchased the following products in the past in order:\n [’0. Scooby-Doo Unmasked - PlayStation 2’, ’1. Scooby-Doo: Mystery Mayhem’, ’2. Thrillville: Off the Rails’, ’3. Pac Man World 2 - PlayStation 2’]\n\n Then if I ask you to recommend a new product to me according to the given purchasing history, you should recommend Charlie and the Chocolate Factory - PlayStation 2 and now that I’ve just purchased Charlie and the Chocolate Factory - PlayStation 2, there are 20 candidate products that I can consider to purchase next:\n [’0. Controller Gear Controller Stand v1.0 - Officially Licensed - Black - Xbox One’, ’1. NHL 08 - PlayStation 2’, ’2. Hyperdimension Neptunia Victory - Playstation 3’, ’3. DOOM [Online Game Code]’, ’4. Sega Dreamcast Controller’, ’5. Mayflash F300 Arcade Fight Stick Joystick for PS4 PS3 XBOX ONE XBOX 360 PC Switch NeoGeo mini’, ’6. Fallout New Vegas Ultimate Edition’, ’7. Star Wars Galactic Battlegrounds: Clone Campaigns (Expansion Pack)’, ’8. Hatsune Miku: Project DIVA X - PlayStation 4’, ’9. GameMaster Sony PS Vita Anti-Bacterial Screen Protector’, ’10. NBA 2K9’, ’11. Dragon Ball Xenoverse - Xbox One’, ’12. Enhanced Thumb Grips Quad Pack BRAND NEW for Playstation Vita’, ’13. Dead Rising 3: Apocalypse Edition’, ’14. Hitman 2 Silent Assassin - Xbox’, ’15. Dracula Resurrection - PC’, ’16. Jekyll and Hyde - PC’, ’17. Thief - Playstation 3’, ’18. MyLifeUNIT Hand Grip Handle Stand for Nintendo New 3DS XL LL’, ’19. Sony PSP-1001K PlayStation Portable (PSP) System (Black)’]\n Please rank these 20 products by measuring the possibilities that I would like to purchase next most, according to the given purchasing records. Please think step by step.\n Please only output the order numbers after ranking. Split these order numbers with line break.*
- **Output:** *1 \n 6 \n 13 \n 7 \n 11 \n 3 \n 14 \n 17 \n 0 \n 2 \n 8 \n 10 \n 18 \n 5 \n 9 \n 15 \n 16 \n 4 \n 12 \n 19*
- **Result:** Ground-Truth *“NHL 08 - PlayStation 2”* **rank 0**