What LLMs Miss in Recommendations: Bridging the Gap with Retrieval-Augmented Collaborative Signals

Shahrooz Pouryousef
UMass Amherst
Amherst, MA, USA
shahrooz@cs.umass.edu

Abstract

User-item interactions contain rich collaborative signals that form the backbone of many successful recommender systems. While recent work has explored the use of large language models (LLMs) for recommendation, it remains unclear whether LLMs can effectively reason over this type of collaborative information. In this paper, we conduct a systematic comparison between LLMs and classical matrix factorization (MF) models to assess LLMs' ability to leverage user-item interaction data. We further introduce a simple retrieval-augmented generation (RAG) method that enhances LLMs by grounding their predictions in structured interaction data. Our experiments reveal that current LLMs often fall short in capturing collaborative patterns inherent to MF models, but that our RAG-based approach substantially improves recommendation quality—highlighting a promising direction for future LLM-based recommenders.

Keywords

LLMs, Recommendation systems, Collaborative information

1 Introduction

Large Language Models (LLMs) [1, 20] have demonstrated impressive capabilities in a wide range of tasks, including reasoning over textual input [8, 14], answering complex questions [9, 18], generating fluent text [7], and encoding world knowledge [17]. As a result, LLMs have been increasingly adopted in diverse research domains such as data collection [21], summarization [3, 13], translation [10, 15, 28], and visualization [11, 23].

More recently, researchers have begun exploring the potential of LLMs in recommender systems, where the goal is to deliver personalized item suggestions that align with a user's preferences—such as recommending movies, products, or articles [2, 5, 16, 24]. This emerging line of work, often referred to as LLMs as recommenders (LLMRec), presents an exciting new research direction that leverages the capabilities of LLMs to tackle core challenges in recommender systems [2, 25, 27]. Within this domain, LLMs have primarily been applied to two key subtasks: item understanding and user modeling. For example, in content-based filtering, LLMs can generate rich item representations from textual descriptions and infer user preferences from their interaction history [12, 22].

LLMs have been utilized in recommender systems (RSs) under two main paradigms. LLMs as Recommenders (LLMs-as-RSs) refers to approaches where LLMs are directly prompted or fine-tuned to function as recommenders. In contrast, LLM-enhanced RSs leverage the knowledge stored in LLM parameters to enhance traditional recommender models—for example, by generating text-based item/user representations or embeddings.

One limitation of LLMRec methods is their insufficient modeling of collaborative information embedded in user-item co-occurrence patterns. To address this, Sun et al. [19] proposed an approach that distills the world knowledge and reasoning capabilities of LLMs into a collaborative filtering recommender system. Their method adopts an in-context, chain-of-thought prompting strategy, focusing on the LLM-enhanced RS paradigm.

However, the integration of collaborative filtering within the LLMs-as-RSs paradigm—where recommendations are derived from patterns across many users—remains relatively underexplored. This gap is due in part to fundamental challenges: it is unclear how to adapt LLMs to effectively reason over collaborative signals, and the input length limitations of LLMs make it difficult to encode dense interaction histories needed to learn from similar users at scale.

In this paper, we take a step toward understanding this gap by analyzing the reasoning capabilities of LLMs over collaborative signals. Our first research question is: Can LLMs capture and reason over collaborative patterns? To assess the basic reasoning ability of LLMs in utilizing collaborative information, we compare their performance with matrix factorization, a classical and widely used method for modeling collaborative information in recommender systems. This comparison allows us to evaluate whether LLMs can reason over collaborative signals at a level comparable to—even if not superior to—a simple yet effective baseline like matrix factorization.

Our second research question investigates how to improve LLMs' ability to reason over collaborative data. To this end and overcome to the challenges that mentioned above, we explore a retrieval-augmented generation (RAG) approach that supplies the LLM with relevant user-item interaction information at inference time, aiming to bridge the gap between LLMs and traditional collaborative filtering methods.

2 Preliminaries

In this study, we explore the reasoning ability of LLMs on collaborative filtering information. Our investigation is centered around a matrix of interactions between users and items, i.e., M. Let $\mathcal{U} = \{u_1, \ldots, u_n\}$ denote the set of all n users and $\mathcal{I} = \{v_1, \ldots, v_m\}$ denote the set of all m items (movies). The matrix $M \in \mathbb{R}^{n \times m}$ represents the rating behavior of users over items, where each row corresponds to a user in $\mathcal{U}$ and each column to an item in $\mathcal{I}$. The entry M_{ij} indicates the rating that user $u_i \in \mathcal{U}$ has given to item $v_j \in \mathcal{I}$, with values typically ranging from 1 (disliked) to 5 (highly liked), or 0 if the user has not rated the item.

To enable large language models (LLMs) to understand collaborative signals in the interaction matrix, it is necessary to provide

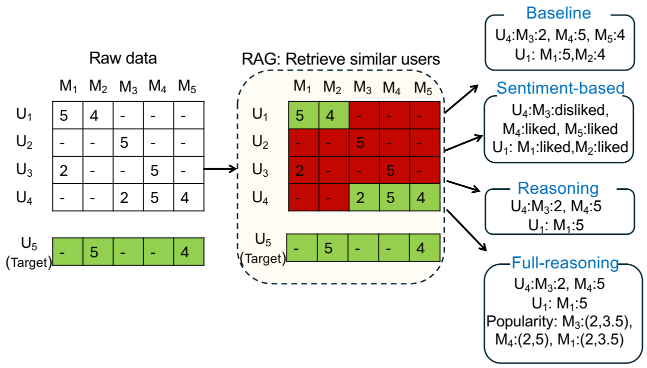

Figure 1: Comparison of four prompt-generation strategies for movie recommendation based on retrieved user–movie similarities. Each method varies in how it incorporates similar users' ratings, handles previously seen movies, and structures the prompt for downstream recommendation.

them with structured information derived from the matrix. However, including the entire matrix in the input is infeasible, especially in large-scale datasets with many users and items. Moreover, presenting all such information in a prompt can overwhelm the LLM, making it difficult to interpret the collaborative structure and respond effectively to recommendation queries.

Recently, retrieval-augmented generation (RAG) has demonstrated impressive results in enhancing LLM performance across a variety of tasks [4, 6, 26]. RAG approaches integrate external retrieval mechanisms to supplement LLMs with relevant and targeted information, thereby improving the accuracy, relevance, and contextual understanding of generated responses.

In the following section, we present our method for applying retrieval-augmented generation (RAG) to enhance the effectiveness and efficiency of LLMs in leveraging collaborative filtering information.

3 Methodology

Given a target user u_t, our goal is to extract relevant signals from the interaction matrix M to enable an LLM to predict and recommend movies that the user is likely to like.

To identify useful information, we first retrieve users most similar to the target user. These similar users are assumed to carry the most informative signals for inferring the preferences of u_t, as their past ratings can help guide the recommendation of unseen items.

To identify users most similar to the target user u_t, we construct a user–item interaction matrix by pivoting the masked dataset (test data are masked for all models) such that each row corresponds to a user and each column to an item, with missing ratings filled with zero. We then compute pairwise user similarities using cosine similarity:

$$\text{sim}(u_t, u) = \frac{M_{u_t} \cdot M_u}{\|M_{u_t}\| \cdot \|M_u\|}$$

We define the set of top-k similar users as:

$$\mathcal{N}_k(u_t) = \text{Top-}k\left(\{\text{sim}(u_t, u) \mid u \in \mathcal{U} \setminus \{u_t\}\}\right)$$

where $\mathcal{U}$ is the set of all users, and $\mathcal{N}_k(u_t) \subset \mathcal{U} \setminus \{u_t\}$ denotes the k most similar users to u_t. For each $u \in \mathcal{N}_k(u_t)$, we define their set of ratings as:

$$R_u = \{(j, M_{u_j}) \mid j \in \mathcal{I},\ M_{u_j} \neq 0\}$$

We use the information from these similar users and their ratings to construct the input for the LLM in our prompt-generation framework. Specifically, the sets R_u for all $u \in \mathcal{N}_k(u_t)$ provide the relevant collaborative signals that are encoded into the prompt. Let $\mathcal{R}_{\mathcal{N}_k(u_t)} = \bigcup_{u \in \mathcal{N}_k(u_t)} R_u$ denote the union of all such rating sets. This set $\mathcal{R}_{\mathcal{N}_k(u_t)} \subset \mathcal{I} \times \mathbb{R}$ represents the pool of item–rating pairs from users most similar to the target user u_t, and serves as a compact, informative context for the LLM.

In the following subsections, we present four distinct strategies for incorporating $\mathcal{R}_{\mathcal{N}_k(u_t)}$ into the prompt. Each strategy varies in how it structures this information, filters relevant ratings, and balances the trade-off between informativeness and prompt length.

3.1 Prompt Generation Strategies

We explore four distinct strategies for constructing our prompts. Each approach presents user rating data differently to investigate its impact on model performance and prompt efficiency.

3.1.1 ***Unfiltered Full Ratings Prompt (Baseline).*** This baseline approach includes the full rating history of all top-k similar users without applying any filtering or deduplication. Each user's ratings are presented in their raw form. While the fraction parameter (f) still controls how much of the similar users' data is included in the prompt, no preference is given to unseen items or highly rated content. This method offers a comprehensive view of all available preferences, potentially enriching the context provided to the LLM.

Notably, we include all sampled ratings from similar users, even for items that the target user has already rated. We refer to this approach as the *baseline.*

Example:

> Target user A has rated the following movies: M1 (5), M2 (3), M3 (4)...
> Top-k similar users have rated:
> User 101 rated: M1 (1), M7 (3), M2 (2)
> User 24 rated: M2 (3), M4 (4)
> ...
> Which 10 movies should user A watch next that they haven't seen?

3.1.2 ***Sentiment-Based Prompt.*** This strategy organizes the ratings from similar users into three sentiment categories: *Liked*, *Neutral*, and *Disliked.* These categories are defined by rating thresholds: ratings ≥ 4 are classified as *Liked*, a rating of 3 as *Neutral*, and ratings ≤ 2 as *Disliked.* As with other strategies, we exclude any movies that the target user has already rated to focus only on unseen content.

Example:

> Target user A has rated the following movies: M1 (5), M2 (3), M3 (4)...
> Top-k similar users to user A have collectively rated the following **unseen** movies:
> **Liked:** 234, 567, 890

> **Neutral:** 111, 222
> **Disliked:** 333, 444
> Based on these patterns, which 10 movies should user A watch next?

3.1.3 ***Reasoning-Based Prompt.*** This strategy builds upon the baseline by introducing two key modifications. First, we remove any movies from the similar users' ratings that have already been seen by the target user, ensuring that all recommended items are truly unseen. Second, we sample a fraction f of the remaining ratings for inclusion in the prompt, controlling the prompt size while preserving informative signals.

Additionally, the prompt is explicitly framed as a reasoning task by appending a directive such as: *"Reason based on the patterns above: which 10 movies should user A watch next that they haven't seen?"* This phrasing encourages the model to engage in logical inference rather than relying purely on pattern matching, potentially improving generalization and recommendation quality.

Example:

> Target user A has rated the following movies: M1 (5), M2 (3), M3 (4)...
> Top-k similar users to user A have collectively rated the following unseen movies:
> User 101 rated: M7 (3), M6 (2)
> User 24 rated: M8 (3), M4 (5)
> Reason based on the patterns above: which 10 movies should user A watch next that they haven't seen?

3.1.4 ***Full Reasoning.*** In the *Reasoning* approach, the LLM only sees data from the top-k similar users, and therefore lacks access to global signals—such as the overall popularity of movies across the user base. To address this limitation, we propose an enhanced variant that augments the prompt with additional popularity-related statistics.

For each movie $v_j \in \mathcal{I}$, we compute two global metrics based on the interaction matrix $M \in \mathbb{R}^{n \times m}$:

- **Rating Count:** The number of users who rated movie v_j:

$$\text{Count}(v_j) = \left|\left\{u_i \in \mathcal{U} \mid M_{ij} \neq 0\right\}\right|$$

- **Average Rating:** The average rating received by movie v_j:

$$\text{AvgRating}(v_j) = \frac{\sum_{u_i \in \mathcal{U}} M_{ij} \cdot \mathbf{1}[M_{ij} \neq 0]}{\text{Count}(v_j)}$$

These statistics are appended to each corresponding movie entry in the "Reasoning" prompt. This allows the LLM to consider not only the preferences of similar users but also the broader appeal and quality of each recommended item.

Example:

> Target user A has rated the following movies: M1 (5), M2 (3), M3 (4)...
> Top-k similar users to user A have collectively rated the following unseen movies:
> User 101 rated: M7 (3), M6 (2)
> User 24 rated: M8 (3), M4 (5)
> Movie popularity stats:
> M4 — Count: 231, AvgRating: 4.2
> M6 — Count: 121, AvgRating: 3.6
> M7 — Count: 298, AvgRating: 3.8
> M8 — Count: 83, AvgRating: 3.2
> Reason based on the patterns above and the popularity statistics: which 10 movies should user A watch next that they haven't seen?

4 Evaluation

4.1 Dataset

We evaluate the effectiveness of using LLMs for personalized movie recommendation using the widely studied MovieLens 100K dataset. This dataset comprises 100, 000 ratings from 943 users on 1, 682 movies. As a preprocessing step, we remap both user and movie IDs to be contiguous integers starting from zero to ensure consistency across model input formats. We sort each user's rating history chronologically and randomly mask 20% of their ratings to simulate unobserved preferences. These masked ratings are withheld and later used to evaluate recommendation accuracy, while the remaining 80% of ratings are retained to simulate user history and compute user-user similarities.

To evaluate the models fairly across different user profiles, we split the users into two groups based on activity level. Users with a number of ratings above the dataset's median are categorized as "hot users," and the rest as "cold users." We randomly sample 350 users from each group.

4.2 Experimental and parameter settings

For each user, we construct a prompt that includes (1) the known ratings from their retained history, and (2) the ratings from the top-k most similar users. These similar users are selected using cosine similarity computed over the masked user-item matrix. To simulate different levels of auxiliary knowledge, we vary the fraction of each similar user's ratings shown in the prompt from 25% to 100%. We use f to indicate the fraction value. We also explicitly inform the LLM about the range of movie IDs in the dataset to provide context.

The prompt concludes with a question asking the model to suggest 10 movies that the target user has not seen. The LLM is then queried using OpenAI's ChatCompletion API, and its response is parsed to extract movie IDs. These predictions are evaluated using NDCG (Normalized Discounted Cumulative Gain) and Hit@10 metrics. NDCG evaluates how well a ranked list puts the most relevant items near the top and Hit@10 metric measures the proportion of ground-truth masked movies that appear in the model's top-10 recommendations. Formally, for each user, Hit@10 is calculated as the cardinality of the intersection between the predicted and held-out movies divided by 10. We compute and average this value across all users in both the hot and cold user groups.

To establish a strong baseline, we also implement a matrix factorization model in PyTorch and train it using stochastic gradient descent over a range of hyperparameters, including different numbers of latent factors ({10, 20, 50, 100}) and batch sizes ({8, 16, 32, 64,128, and 256}). The model is trained for up to 3, 000 epochs or until convergence, with performance evaluated on a held-out test set. We report the best-performing configuration after convergence based on the highest NDCG score for the testing cases. The model is trained on the same 80% unmasked ratings used in the LLM setup.

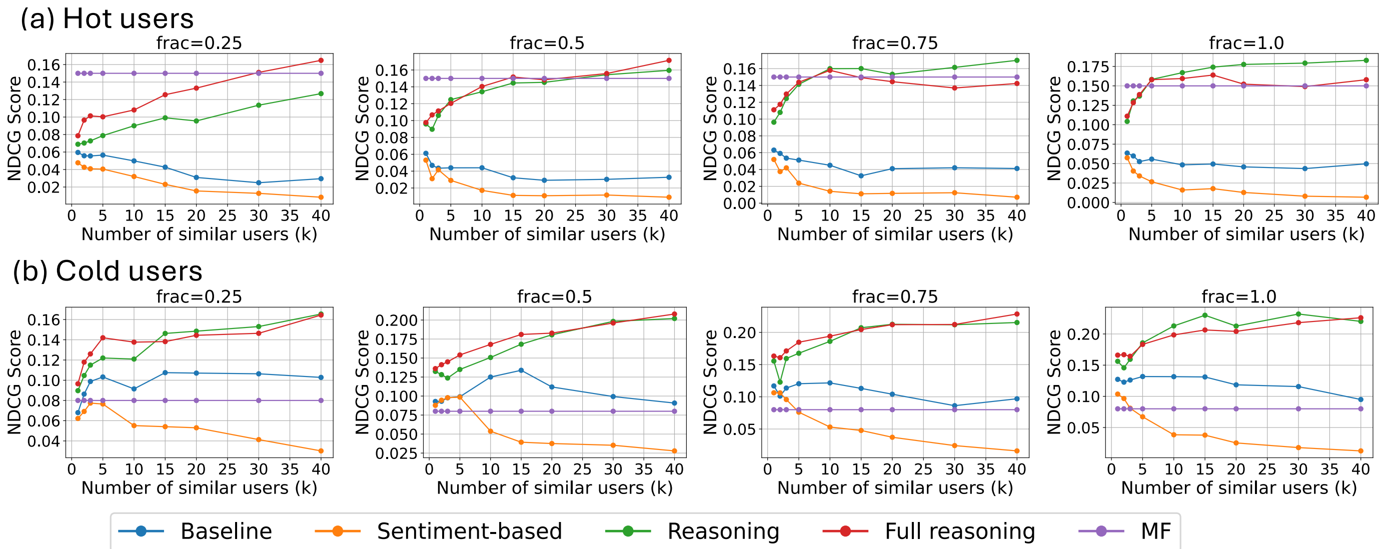

Figure 2: NDCG score for different prompt generation strategies and MF as a function of number of similar users (k) and different percentage of their information in the prompt (f). Number of users is 350 in both cold and hot groups. Top is the hot users and bottom is cold users.

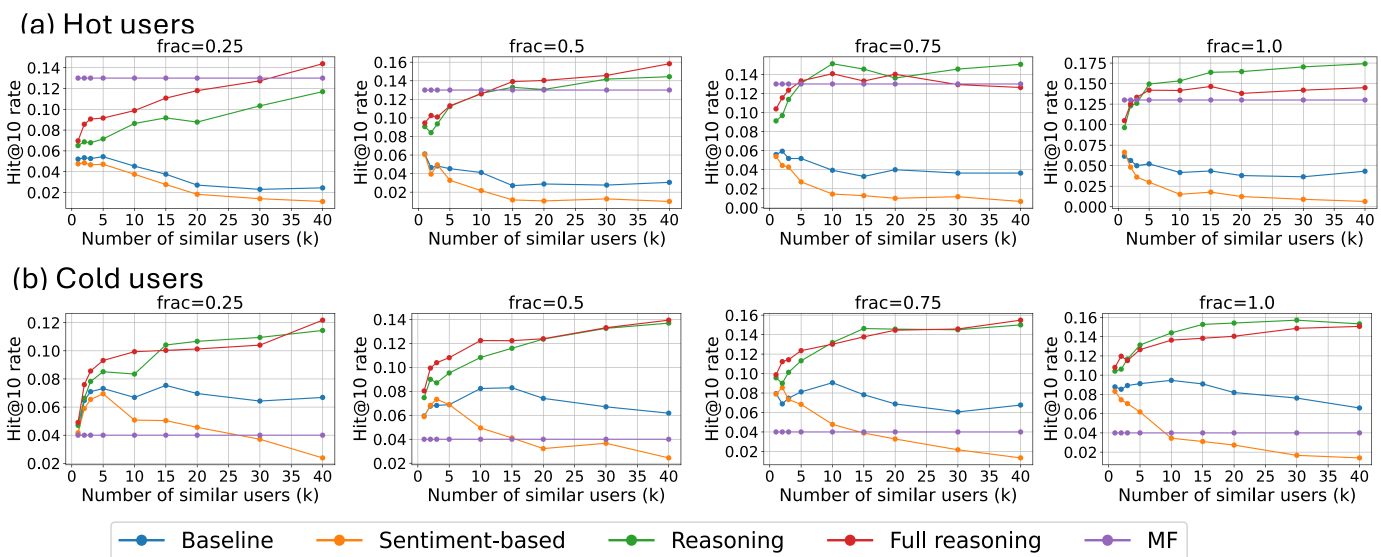

Figure 3: Hit@10 score for different prompt generation strategies and MF as a function of number of similar users (k) and different percentage of their information in the prompt (f). Number of users is 350 in both cold and hot groups. Top is the hot users and bottom is cold users.

4.3 Results and Discussion

Figure 2 presents the average NDCG scores obtained across different prompt generation strategies as a function of the number of similar users k, for multiple fractions of their rating histories (f). We observe that reasoning-based prompts (such as reasoning and full reasoning) consistently improve in performance as k increases. This suggests that LLMs benefit from richer collaborative information when it is presented in a structured and interpretable format. In contrast, simpler strategies such as sentiment-based or original tend to decrease in NDCG as k increases. This likely occurs because these prompts include more raw data without structure

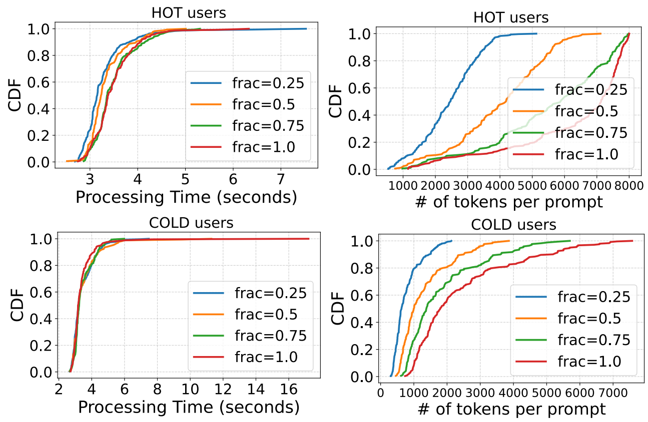

Figure 4: Processing time of prompts and the number of tokens in each prompt for hot and cold users.

or abstraction, making it harder for the LLM to extract relevant signals—highlighting the importance of structured prompt engineering. These results support integrating techniques like retrieval-augmented generation (RAG) or knowledge graphs to organize collaborative user–item interactions more effectively.

As we increase the fraction of similar user data included in the prompt, we observe a general increase in overall NDCG scores across reasoning strategies. This aligns with the expectation that providing more evidence to the LLM leads to stronger predictions. However, an interesting trend appears when comparing cold vs. hot users: LLMs tend to perform better on cold users than hot users, while Matrix Factorization (MF) exhibits the opposite behavior—achieving higher scores for hot users and lower for cold ones. This is expected, as MF relies on sufficient user-item interactions to learn embeddings, making it more effective for users with rich histories.

For LLMs, one possible explanation for stronger performance on cold users is reduced confusion due to fewer input ratings. Additionally, cold users typically have fewer masked (unseen) movies—sometimes only 2 or 3—since we mask 20% of each user's ratings. When the LLM is asked to recommend 10 movies, the probability of hitting those few masked movies is relatively high. In contrast, hot users often have 100+ ratings, so 20% masking can result in 20–30 unseen movies. Consequently, the LLM has a lower chance of selecting the correct ones, even when reasoning is good, due to the larger candidate set.

Figure 4 shows the cumulative distribution function (CDF) of prompt processing times for cold and hot users using the reasoning_ranked prompt generation strategy, with the number of similar users fixed at $k = 10$, and varying the fraction of each similar user's rating history included in the prompt. By processing time, we refer to the total latency involved in communicating with the GPT server, including the time to transmit the prompt, the model's internal computation, and the time required to receive the response back to the local machine.

For cold users, the CDF curves for different fraction values (e.g., 0.25, 0.5, 0.75, 1.0) are tightly clustered and exhibit nearly identical processing times. This is expected because cold users, by definition, have rated very few movies. Consequently, even when using the full fraction of ratings from their similar users, the total number of ratings included in the prompt remains small. Thus, the prompt length—and hence the request latency—does not vary significantly with the fraction parameter.

In contrast, for hot users, we observe that higher fraction values lead to noticeably longer processing times. Since hot users have many ratings, each similar user is likely to contribute more movie–rating pairs as the fraction increases. This results in significantly longer prompts, which take longer to serialize, transmit to the model, and process on the server. The effect is visible in the rightward shift of the CDF curves for higher fractions.

This analysis highlights that prompt length has a strong effect on processing latency, especially for users with rich interaction histories. It also supports the importance of designing token-efficient prompt generation strategies when scaling LLM-based recommendation systems, particularly for hot users with high k and large data fractions.

5 Conclusion

In the LLMs-as-recommender-systems (LLMs-as-RCs) paradigm, a key challenge is enabling LLMs to effectively incorporate collaborative information. In this paper, we began by analyzing the performance of LLMs in capturing collaborative signals for movie recommendation. We showed that a naive approach—embedding all user information directly into the prompt—makes it difficult for LLMs to interpret these signals, often performing worse than simple baselines such as matrix factorization.

To address this, we proposed a retrieval-augmented generation (RAG) based approach with improved prompting strategies. Our results demonstrate that presenting collaborative signals in a compact format and prompting the LLM to reason over them improves recommendation performance compared to traditional baselines.

Moreover, our method is both token-efficient and effective, achieving better results on standard evaluation metrics while minimizing prompt length.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Keqin Bao, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. 2023. Tallrec: An effective and efficient tuning framework to align large language model with recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 1007–1014.

[3] Lochan Basyal and Mihir Sanghvi. 2023. Text summarization using large language models: a comparative study of mpt-7b-instruct, falcon-7b-instruct, and openai chat-gpt models. *arXiv preprint arXiv:2310.10449* (2023).

[4] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 17754–17762.

[5] Sunhao Dai, Ninglu Shao, Haiyuan Zhao, Weijie Yu, Zihua Si, Chen Xu, Zhongxiang Sun, Xiao Zhang, and Jun Xu. 2023. Uncovering chatgpt's capabilities in recommender systems. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 1126–1132.

[6] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6491–6501.

[7] Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. 2023. Enabling large language models to generate text with citations. *arXiv preprint arXiv:2305.14627*

(2023).

[8] Jie Huang and Kevin Chen-Chuan Chang. 2022. Towards reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10403* (2022).

[9] Zhengbao Jiang, Jun Araki, Haibo Ding, and Graham Neubig. 2021. How can we know when language models know? on the calibration of language models for question answering. *Transactions of the Association for Computational Linguistics* 9 (2021), 962–977.

[10] Tom Kocmi, Eleftherios Avramidis, Rachel Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark Fishel, Markus Freitag, Thamme Gowda, Roman Grundkiewicz, et al. 2024. Findings of the WMT24 general machine translation shared task: the LLM era is here but mt is not solved yet. In *Proceedings of the Ninth Conference on Machine Translation*. 1–46.

[11] Guozheng Li, Xinyu Wang, Gerile Aodeng, Shunyuan Zheng, Yu Zhang, Chuangxin Ou, Song Wang, and Chi Harold Liu. 2024. Visualization generation with large language models: An evaluation. *arXiv preprint arXiv:2401.11255* (2024).

[12] Jianghao Lin, Xinyi Dai, Yunjia Xi, Weiwen Liu, Bo Chen, Hao Zhang, Yong Liu, Chuhan Wu, Xiangyang Li, Chenxu Zhu, et al. 2025. How can recommender systems benefit from large language models: A survey. *ACM Transactions on Information Systems* 43, 2 (2025), 1–47.

[13] Yixin Liu, Kejian Shi, Katherine S He, Longtian Ye, Alexander R Fabbri, Pengfei Liu, Dragomir Radev, and Arman Cohan. 2023. On learning to summarize with large language models as references. *arXiv preprint arXiv:2305.14239* (2023).

[14] Aske Plaat, Annie Wong, Suzan Verberne, Joost Broekens, Niki van Stein, and Thomas Back. 2024. Reasoning with large language models, a survey. *arXiv preprint arXiv:2407.11511* (2024).

[15] Vikas Raunak, Amr Sharaf, Yiren Wang, Hany Hassan Awadallah, and Arul Menezes. 2023. Leveraging GPT-4 for automatic translation post-editing. *arXiv preprint arXiv:2305.14878* (2023).

[16] Scott Sanner, Krisztian Balog, Filip Radlinski, Ben Wedin, and Lucas Dixon. 2023. Large language models are competitive near cold-start recommenders for language-and item-based preferences. In *Proceedings of the 17th ACM conference on recommender systems*. 890–896.

[17] Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. 2023. Large language models encode clinical knowledge. *Nature* 620, 7972 (2023), 172–180.

[18] Karan Singhal, Tao Tu, Juraj Gottweis, Rory Sayres, Ellery Wulczyn, Mohamed Amin, Le Hou, Kevin Clark, Stephen R Pfohl, Heather Cole-Lewis, et al. 2025. Toward expert-level medical question answering with large language models. *Nature Medicine* (2025), 1–8.

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

[20] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[21] Wei Wei, Xubin Ren, Jiabin Tang, Qinyong Wang, Lixin Su, Suqi Cheng, Junfeng Wang, Dawei Yin, and Chao Huang. 2024. Llmrec: Large language models with graph augmentation for recommendation. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 806–815.

[22] Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, Chuan Qin, Chen Zhu, Hengshu Zhu, Qi Liu, et al. 2024. A survey on large language models for recommendation. *World Wide Web* 27, 5 (2024), 60.

[23] Yang Wu, Yao Wan, Hongyu Zhang, Yulei Sui, Wucai Wei, Wei Zhao, Guandong Xu, and Hai Jin. 2024. Automated data visualization from natural language via large language models: An exploratory study. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–28.

[24] Zhenrui Yue, Sara Rabhi, Gabriel de Souza Pereira Moreira, Dong Wang, and Even Oldridge. 2023. Llamarec: Two-stage recommendation using large language models for ranking. *arXiv preprint arXiv:2311.02089* (2023).

[25] Jizhi Zhang, Keqin Bao, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. 2023. Is chatgpt fair for recommendation? evaluating fairness in large language model recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 993–999.

[26] Taolin Zhang, Dongyang Li, Qizhou Chen, Chengyu Wang, Longtao Huang, Hui Xue, Xiaofeng He, and Jun Huang. 2024. R 4: Reinforced Retriever-Reorder-Responder for Retrieval-Augmented Large Language Models. In *ECAI 2024*. IOS Press, 2314–2321.

[27] Zihuai Zhao, Wenqi Fan, Jiatong Li, Yunqing Liu, Xiaowei Mei, Yiqi Wang, Zhen Wen, Fei Wang, Xiangyu Zhao, Jiliang Tang, et al. 2024. Recommender systems in the era of large language models (llms). *IEEE Transactions on Knowledge and Data Engineering* (2024).

[28] Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. 2023. Multilingual machine translation with large language models: Empirical results and analysis. *arXiv preprint arXiv:2304.04675* (2023).

Received 20 February 2024; revised 12 March 2024; accepted 5 June 2024