

# Evaluation of Refined Conversational Recommendation Based on Reprompting ChatGPT with Feedback

Kyle D. Spurlock\*  
kyle.spurlock@louisville.edu  
University of Louisville  
Louisville, Kentucky, USA

Esin Saka  
esinsaka@gmail.com  
Microsoft  
Seattle, Washington, USA

Cagla Acun  
caglacun@gmail.com  
University of Louisville  
Louisville, Kentucky, USA

Olfa Nasraoui  
olfa.nasraoui@louisville.edu  
University of Louisville  
Louisville, Kentucky, USA

## ABSTRACT

Recommendation algorithms seldom consider direct user input, resulting in superficial interaction despite efforts to include the user through conversation. Recently, Large Language Models (LLMs) have gained popularity across a number of domains for their extensive knowledge and transfer learning capabilities. For instance, ChatGPT boasts impressive interactivity and an easy-to-use interface. In this paper, we investigate the effectiveness of ChatGPT as a top-N conversational recommendation system. We build a rigorous evaluation pipeline to simulate how a user might realistically probe the model for recommendations: by first instructing and then reprompting with feedback to refine the recommendations. We further explore the effect of popularity bias in ChatGPT’s recommendations, and compare its performance to baseline recommendation models. We find that reprompting with feedback is an effective strategy to improve recommendation relevancy, and that popularity bias can be mitigated through prompt engineering. Code and data are available at <https://github.com/kspurlock/Reprompting4Rec>.

## CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Retrieval effectiveness**; **Language models**.

## KEYWORDS

Recommender Systems, Generative AI, LLM, Prompt Engineering

### ACM Reference Format:

Kyle D. Spurlock, Cagla Acun, Esin Saka, and Olfa Nasraoui. 2024. Evaluation of Refined Conversational Recommendation Based on Reprompting ChatGPT with Feedback. In *Proceedings of Workshop on Generative AI for Recommender Systems and Personalization (GenAIRecP’24)*. ACM, New York, NY, USA, 10 pages. <https://doi.org/XXXXXXX.XXXXXXX>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

GenAIRecP’24, August 25, 2024, Barcelona, Spain

© 2024 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06...\$15.00

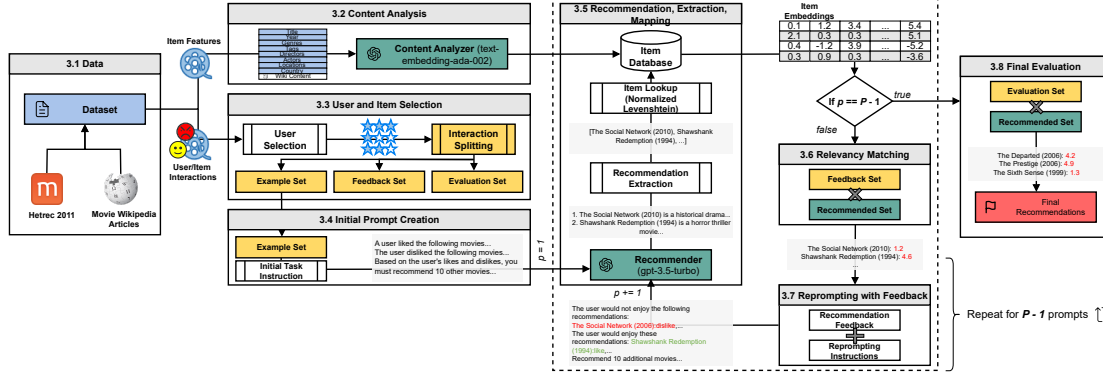
<https://doi.org/XXXXXXX.XXXXXXX>

## 1 INTRODUCTION

State-of-the-art (SOTA) recommender systems rely on Machine Learning (ML) algorithms to predict human behavior and interests based on previous activities, behavior, or direct feedback. However, in most recommendation algorithms implemented in practice, there is no direct communication between the human and the model. One may not want to argue with an AI about their preferences when trying to choose a new movie to watch or a book to read, but the possibility to do so presents an interesting opportunity for a model to learn directly from the user it serves in real time.

The task of introducing interactivity into recommendation through conversational recommender systems [25] has been addressed in the past with varying degrees of success. When natural language processing was far less advanced, chatbots like [20] were able to tailor the recommendation experience to the user; albeit restricted in their response. More recently, techniques that use deep learning have become far more interesting to engage with. The encapsulation of Large Language Models (LLM) like ChatGPT into a non-technical, user-friendly interface have further redefined what it means to use and interact with AI systems. This work is interested in ChatGPT specifically because of this ease of access, with the assumption that anyone could go to its interface and use it for recommendation with little effort.

In this work we focus on evaluating the conversational top-N recommendation potential of the OpenAI’s GPT-3.5 based ChatGPT [15]. Recent works [5, 10, 12] have largely evaluated ChatGPT’s recommendation potential on the basis of single inputs and outputs, neglecting its ability to converse. This format often requires the model to choose an option out of a predetermined set that best completes the task; which we argue is not representative of how a user would actually interact with such a system for a task like recommendation. We choose to structure our study in a way that utilizes the conversational ability of ChatGPT as part of the recommendation process. The goal is to evaluate how ChatGPT performs at recommendation in a more natural setting, i.e. how can it generate pertinent answers to the task that have not been provided beforehand. Furthermore, as LLMs have consumed enormous swathes of data in training and some recommendable items may appear more often than others, we are interested in determining whether the model exhibits popularity bias in its recommendations [3].



**Figure 1: Proposed pipeline for evaluating the effect of conversation in recommendation.**  $P$ =number of prompts,  $p$ =prompt number. Each section corresponds to the section of the same name in the methodology.

## 2 RELATED WORK

### 2.1 Prompt Engineering

Prompt engineering is an emergent field of study that centers around LLMs’ ability to reason on a task only through information provided “in context,” without specific fine-tuning for that task. Methods and techniques in this field aim to explore and optimize communication patterns in order to produce the most desirable outcomes while performing no updates on model parameters. Some of the most notable prompt engineering patterns include **Zero-shot** [1]: the model is provided with only instructions and asked to complete a task; **Few-shot** [1]: the model is given examples demonstrating a task, and is then asked to repeat this task by generating its own output for a similarly structured question. **One-shot** specifies that the prompt contains *one* example; **Chain-of-Thought (CoT) prompting**: the model is gradually asked to produce intermediate answers before giving the final answer to a multi-step problem [22]. The idea is to mimic an intuitive multi-step thought process when working through a reasoning problem. Most other approaches can be derived from these archetypes and either vary the amount of information or present the task in a different way. Other advanced prompting methods use knowledge graphs [13].

### 2.2 Language Models as Recommenders

The extensive domain knowledge encapsulated in LLMs have made them of keen interest to the recommendation community. Early approaches concentrated predominantly on single input-output tasks. For example, BERT4Rec [19] refines the encoder-only transformer model BERT [4], and applies it to sequential recommendation using item IDs; achieving notable gains over previous RNN-led benchmarks. LMRecSys [26] was among the first methods to explore in-context sequential recommendation with LLMs, showcasing the performance of BERT and GPT-2 [16] across a variety of zero-shot prompts. Another study [10] followed this with more recent models, with findings that show the benefit of using an LLM for sequential recommendation with fine-tuning. [12] underscored the general-purpose recommendation potential of ChatGPT. It emphasized the power of prompt engineering, converting recommendation tasks into natural language tasks, and assessing the model’s performance

without explicit fine-tuning. Authors in [5] further evaluated GPT-3 models as *augmented* LMs to interface with existing recommendation systems to perform post-processing of their outputs. There is further similar work in [17], where authors explore the potential of LLMs in making recommendations based on item-based and language-based preferences. Other work includes [14] and methods surveyed in [7, 21]. **The main difference between existing work and ours is the explicit focus on the real-time conversational dynamics of an LLM like ChatGPT for item-based recommendation.**

## 3 METHODOLOGY

For clarity when reading the following sections, “ChatGPT” refers specifically to the model *gpt-3.5-turbo-0125* which was the latest model available during the bulk of this study, and the model serving the public chat interface. **Additionally, for more details on some parts of the system, we refer the reader to the Appendix. We present the total parameterization of the system in the Appendix Table A.1 for ease of reference.**

### 3.1 Data

Our study is built around using the HetRec2011 dataset [2] as the ground truth for evaluation. HetRec2011 is an extended version of the MovieLens10M [6] dataset containing additional film information sourced from IMDB<sup>1</sup> and Rotten Tomatoes<sup>2</sup>. Each film contains the following attributes: title, release year, genres, tags, directors, actors, filming locations, and country of origin. The information available for each movie is further expanded by crawling its Wikipedia page; and we successfully retrieve the articles for 9,722 out of 10,197 movies in HetRec2011 as of 8/1/2023.

### 3.2 Content Analysis

Our objective is to assess ChatGPT’s natural output without providing candidate items ahead of time; which is challenging due to its closed system with no direct access to ground truth information.

<sup>1</sup><https://www.imdb.com/>

<sup>2</sup><https://www.rottentomatoes.com/>

Thus, we utilize OpenAI's *text-ada-embedding-002* to generate continuous representations for movies in HetRec2011, leveraging their shared GPT-3 foundation. However, gauging exact information learned by LLMs from vast data sources remains complex. While basic descriptors (title, release year, genres) might suffice if a movie has abundant related information in the training data, rarer movies could require more information to build a robust representation. Because of this, we generate embeddings for HetRec2011 items and vary the content depth for each, as described below, according to four *content levels*.

Level 1 content contains only the most basic information about a movie. Level 2 incorporates the extra movie attributes from HetRec2011. Level 3 contains additional text scraped from Wikipedia, and level 4 contains level 3 content but with the top 5% most frequent word-level tokens and stop words removed. This last level is produced based on the notion that frequently occurring words are not meaningful for determining similarity [11]. We provide an in-depth experiment on the effect content level plays on the system in Appendix B, and ultimately choose content level 4 for further experiments.

### 3.3 User and Item selection

A sample of 50 users was taken between the 50th and 75th percentile of item interactions, corresponding to a minimum of 122 total interactions with at least 30 negative interactions. As long as the constraints are satisfied, the choice of the user is considered arbitrary. It is only necessary that a user has accumulated enough interactions to properly estimate their preferences. The motivating factors for this small sample size were the cost to use OpenAI's API, as well as ChatGPT's inherently non-deterministic output even while using a recently introduced seed parameter. For the latter case, we prefer to replicate the runs to minimize variance.

Given the set of users  $U$  and the set of items  $I$ , each user  $u \in U$  has a set of interactions  $R_u$ . Each interaction  $r_{ui} \in R_u$  is given as a rating on a Likert scale of [1, 5], and indicates the user's sentiment towards an item  $i \in I$ . Interactions are considered positive when  $r_{ui} \geq 3$ , and negative otherwise.  $R_u$  is split into three random subsets  $E_u$ ,  $F_u$ ,  $T_u$ ; with each split serving a different purpose throughout the pipeline. The splits are further stratified by rating for weighting purposes when estimating recommended item relevancy.

The items in the **example set**  $E_u$  are used in the initial prompt construction to provide ChatGPT with preliminary information on a user's preferences. The **feedback set**  $F_u$  is used solely during the *reprompting* stage to help further develop ChatGPT's understanding of the user profile over several iterations. Lastly, the **evaluation set**  $T_u$  is used to evaluate the final set of recommendations. The importance of dividing the interactions amongst these subsets is to separate the information ChatGPT is allowed to learn from versus what it will be tested on.

### 3.4 Initial Prompt Creation

The initial prompt that ChatGPT receives should inform it of its task and optionally provide additional information that helps it to complete this task favorably. For our purposes, we wish to paint an initial picture of a user's tastes that ChatGPT is then able to

produce recommendations off of. We experiment with three different prompting strategies, shown in-detail in Figure E.1 of the Appendix.

Items from the set of examples  $E_u$  are injected into each prompt and appended with an identifier indicating whether an item was liked or disliked. Example items are followed by instructional text that requests specific behavior from the model, such as ranking by confidence. Other parameters are injected into the prompt that indicates the number of recommendations requested,  $k$ , and a constraint on the recommendation space *release\_cutoff*. We specify the latter to avoid making recommendations for items not contained in our dataset<sup>3</sup>. For HetRec2011, the most recent movie was released in 2011.

While zero-shot would be the most natural means of prompting for the user, for the sake of comparison we include additional prompting styles with examples of the task in one-shot and Chain-of-Thought (CoT) [22]. We include the latter two options based on findings that prompts displaying higher reasoning have shown an increase in the model's reasoning ability in turn [1, 10, 12, 23]. Both one-shot and CoT options include injected synthetic examples and extra reasoning steps generated through performing similarity search on random examples drawn from the set of items.

### 3.5 Recommendation, Extraction, and Mapping

Once ChatGPT has produced its completion at a given prompting stage, the recommended titles must be extracted. The output is natural language, thus some minor errors in formatting, spelling, and grammar are to be expected. In other studies [5, 12], this has been addressed as a possible issue when extracting and evaluating the recommendations. There were no issues with noncompliance when requesting recommendations as a numbered list, thereby making it simple to extract titles with regular expressions.

Extracted titles are matched to a generated embedding that has been stored in a database. To avoid issues with an exact lookup, the title match is performed using Normalized Levenshtein Similarity (NLS)[24] as a fuzzy lookup approach. NLS weighs the number of single-character transformations required to turn title  $X$  into title  $Y$  proportionally to their lengths, and produces a score in the interval [0, 1]. If title  $X$  is similar enough to title  $Y$  by a parameter *title\_threshold*, we match  $X$  to  $Y$  and return  $Y$ 's embedding.

Out-of-dataset items that cannot be matched with NLS are excluded from evaluation metric computations and the feedback process. This neither penalizes nor rewards the model. To minimize the amount of failed matches, we keep a short list of 363 additional titles that were frequently mismatched over 2,000+ preliminary trials. Titles in this list have their content expanded following the procedure in §3.2. We further monitor what fraction of titles are unmatchable with the **Unmatched Ratio** metric described in §3.8 with more details described in Appendix C.

### 3.6 Relevancy Matching

The most important part of the pipeline is estimating whether a given user would respond positively to a recommended item. To start, a user  $u$  has a split of interactions  $S_u$  where  $S_u := F_u$  when in the reprompting stage, or  $S_u := T_u$  when in the final evaluation

<sup>3</sup>gpt-3.5-turbo-0125 has knowledge up to September 2021 at the time of this research.

**Table 1: Mean metric values for different ChatGPT configurations in the pipeline. Scores are based on a final set of  $k = 20$  recommendations matched against the evaluation set. Best results are colored in each column.**

Prompt	$k$	$P$	Prec (%)	nDCG (%)	ILS (%)	MAP (%)	UR (%)
Without Reprompting							
Zero	20	1	58.6	61.8	79.1	65.0	6e-4
One	20	1	59.7	62.6	79.2	65.8	6e-4
CoT	20	1	52.0	52.7	<b>74.3</b>	53.4	1e-2
With Reprompting							
Zero	5	3	59.3	62.4	79.0	64.7	5e-4
	5	5	61.1	62.4	78.8	63.8	1e-3
	10	3	61.2	65.3	78.9	<b>68.2</b>	1e-3
	10	5	<b>63.7</b>	<b>65.6</b>	79.1	67.4	1e-3
One	5	3	61.6	64.8	78.8	67.6	9e-4
	5	5	61.2	62.0	78.3	63.2	2e-3
	10	3	62.9	64.5	78.7	66.4	1e-3
	10	5	62.7	64.5	78.8	66.0	5e-3
CoT	5	3	61.5	62.8	78.4	64.4	3e-3
	5	5	60.2	62.0	78.7	63.8	2e-3
	10	3	60.7	63.2	78.6	65.1	6e-3
	10	5	62.5	63.7	78.3	65.3	1e-2

stage. The recommender produces a set of recommendations  $\hat{R}_u^{(p)}$  at each prompt  $p$  for  $P$  total prompts. With inspiration from Sarwar et al. [18], we estimate the rating  $\hat{r}_{uj}$  for recommended item  $j \in \hat{R}_u^{(p)}$  by computing a weighted sum of ratings  $r_{ui}$  for items  $i \in S_u$  by:

$$\hat{r}_{uj} = \frac{\sum_{i \in S_u} r_{ui} \cdot \text{sim}(\vec{v}_i, \vec{v}_j) \cdot \mathbb{1}_q(i, j)}{\sum_{j \in S_u} \text{sim}(\vec{v}_i, \vec{v}_j) \cdot \mathbb{1}_q(i, j)} \quad (1)$$

$$\mathbb{1}_q(i, j) = \begin{cases} 1, & \text{sim}(\vec{v}_i, \vec{v}_j) \geq \epsilon_{q_i} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Where  $\text{sim}(\vec{v}_i, \vec{v}_j)$  is the cosine similarity between item embeddings  $\vec{v}_i, \vec{v}_j$ , and  $\epsilon_{q_i}$  is the  $q^{\text{th}}$  quantile of similarities between item  $i$  and all other items in the dataset. If  $\hat{r}_{uj} \geq 3$ , we accept the recommended item  $j$  as a relevant to  $u$ . This by-item similarity threshold is used to combat the high-dimensionality of item representations generated by *text-ada-embedding-002* (with dimensionality  $d = 1536$ ), which skews the similarity distribution. Intuitively, the threshold considers that there is a finite number of reasonably similar items to the recommended item.

### 3.7 Reprompting with Feedback

The reprompting stage involves performing relevancy matching against recommended items, and merely informing ChatGPT which of these recommendations were good or bad. We make one small addition to the instructions to ask ChatGPT to avoid making duplicate recommendations. This is to aid in exploring the user’s interaction space. In preparing for evaluation, the value for  $k$  is substituted by the value for the final number of recommendations  $k_f$ , and extra context is added to the next prompt. Both of these prompts can be seen in Figure E.2 in the Appendix.

**Table 2: Comparison of two best pipeline parameterizations (with/without reprompting) against baseline models. Metric values are averaged.**

Model	Prec	nDCG	ILS	MAP	UR
GPT3.5 (Z, k=10, p=5)	<b>63.7</b>	<b>65.6</b>	79.1	<b>67.4</b>	1e-3
GPT3.5 (O, k=20, p=1)	59.7	62.6	79.2	65.8	6e-4
NMF-item (GPT3.5 embed)	26.3	26.2	77.3	25.9	-
NMF-user (GPT3.5 embed)	27.0	28.5	77.0	30.0	-
Random	24.3	24.6	<b>76.0</b>	24.6	-
NMF-item (learned)	62.6	62.9	63.9	63.4	-
NMF-user (learned)	64.6	64.8	63.7	64.7	-

### 3.8 Evaluation of Recommendations

Throughout a conversation with  $P$  prompts,  $k$  recommendations per prompt, and  $k_f$  final recommendations,  $k(P - 1) + k_f$  recommendations in total are generated. We focus evaluation on the final  $k_f$  set of recommendations to compare against the evaluation set  $T_u$ , as the purpose of a recommender should be to filter a large collection of items down to a relevant subset.

Several standard metrics are used to assess overall recommendation quality. **Precision** is used to measure the proportion of ChatGPT’s recommendations that would be relevant to the user. **nDCG** and **MAP** are used to measure ranking performance. **Intralist Similarity (ILS)** [8] measures diversity in the recommendation list. **Novelty** is considered the inverse of an item’s popularity [9], and measures the relative “rareness” of recommendations made for a user. Recall is also measured in some experiments to measure the proportion of interactions that ChatGPT is able to match for a user.

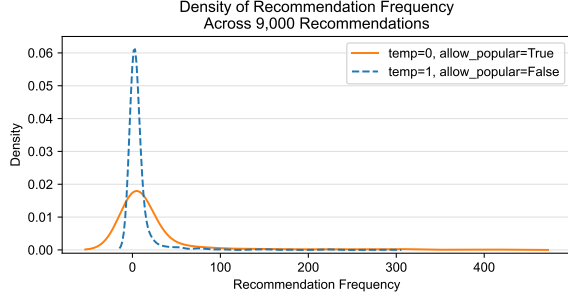
We define a new metric the **Unmatched Ratio (UR)** to measure the fraction of unmatched recommendations made for a user throughout a conversation. The purpose of this metric is to validate the pipeline by ensuring that other metrics are not being significantly skewed from an inability to match ChatGPT’s recommendations to a database item.

For brevity, the formulations for these metrics as they are used in this work are provided in Appendix C.

## 4 EXPERIMENTS

We conduct experiments that aim to answer the following research questions: **RQ1**: How does the ability to converse impact recommendation in large language models? **RQ2**: How do large language models perform at recommendation in their *typical* use-case (as primarily item-based, top-N recommenders)? **RQ3**: Does ChatGPT exhibit popularity bias in recommendation? **Additionally, for more details, we refer the reader to the Appendix sections.**

We utilize a Randomized Complete Block Design (RCBD) to account for the variance in the responses induced by different users and different-sized interaction sets. We perform a block on the user and complete 3 full replicates for each block at each level for the independent variables of interest. Unless otherwise specified, we use an alpha level of  $\alpha = 0.05$  to determine statistical significance. ANOVA is used as a parametric test followed by TukeyHSD. Kruskal-Wallis is used as a non-parametric test followed by Dunn’s test.



**Figure 2: Densities of item recommendation frequencies for two ChatGPT configurations. By setting *temperature* = 1 and *allow\_popular* = False, there is a higher concentration (density) of items recommended with lower frequency. Both configuration have base parameters  $P = 5$ ,  $k = 10$ , *prompt\_style* = ‘zero’.**

We hold the following parameters constant unless specified:  $k_f = 20$ , *example\_size* = 10, *eval\_size* = 0.33, *title\_threshold* = 0.75,  $q = 0.99$ , *temperature* = 0, *random\_state* = 22222.

#### 4.1 Analysis of Iterative Feedback

For this experiment, we aim to answer **RQ1: How does the ability to converse impact recommendation in language models?**. To accomplish this, we wish to compare different parameterizations of ChatGPT with reprompting against direct recommendation.

Our factors of interest are *prompt\_style*,  $k$  (the number of recommendations per prompt), and  $P$  (the number of prompts). Factors  $k$  and  $P$  are combined into a single factor *config* to compare between direct/conversational settings. A two-way ANOVA is used to test for significance in the metrics, for all metrics besides ILS; where Kruskal-Wallis is used instead. We find that all factors are significant, as well as the interaction between *prompt\_style* and *config*. TukeyHSD indicated that the configurations that allow for the most recommendations with  $k = 10$  were significantly different from other levels for precision, nDCG, and MAP. One-shot and zero-shot settings are not significantly different, but both perform better than our CoT prompting across the board. Dunn’s test results for ILS showed that reprompting noticeably increases the similarity between recommendations as the number of prompts increases. CoT prompting significantly decreases ILS, which may indicate that the model fixates on the items initially chosen as examples.

Interactions only show a notable difference between precision configurations, but **this provides evidence that reprompting is effective in making the model’s final recommendations more relevant. With this finding, we answer RQ1.** Aggregation of the mean metric values for each possible configuration in Table 1 shows that the configuration  $k = 10$ ,  $P = 5$ , and *prompt\_style* = ‘zero’ is the best parameterization overall based on raw values. We further examine this model in Appendix Figure F.1 to see how precision varies with coverage across prompts. We see that the model is able to match more relevant items to the user as  $p$  increases, which also shows an increase in precision.

**Table 3: Mean metric values for combinations of *temp* and *allow\_popular* (abbr. *ap*) parameters. *ap* is a boolean parameter taking values (*T*)*true* and (*F*)*alse*.**

<i>temp</i>	<i>ap</i>	Prec	nDCG	ILS	MAP	Novelty	UR
0.0	<i>F</i>	43.0	43.8	72.5	44.8	33.8	3e-2
	<i>T</i>	63.7	<b>65.6</b>	79.1	<b>67.4</b>	28.2	1e-3
0.5	<i>F</i>	43.7	44.4	74.7	45.4	40.7	3e-2
	<i>T</i>	<b>64.0</b>	<b>65.6</b>	79.2	67.0	28.8	2e-3
1.0	<i>F</i>	41.3	42.2	<b>71.2</b>	43.0	<b>58.1</b>	6e-2
	<i>T</i>	62.6	64.3	78.8	65.0	35.7	5e-3

#### 4.2 Analysis of ChatGPT as a Top-N Recommender

We perform a comparison between ChatGPT in the pipeline versus baseline models in order to answer **RQ2: How do language models perform at recommendation in their typical use-case? (as primarily item-based, top-N recommenders)**. The two best parameterizations of the pipeline with and without reprompting, as indicated in Table 1, are selected to represent ChatGPT. We employ four total configurations that utilize Non-negative Matrix Factorization (NMF) as the underlying recommender component. NMF-item produces  $k_f$  recommendations based on aggregate item-item similarity. NMF-user produces  $k_f$  recommendations based on the top-N items found from user-item similarity. The specifics of this process are outlined in Appendix D.

The mean metric values for each model tested can be seen in Table 2. Through post-hoc tests, we find that ChatGPT is significantly better than the Random baseline, which indicates that it is using the knowledge of the user to its advantage. The NMF recommenders evaluated with GPT-3 embeddings perform poorly as expected, but still perform better than random. Interestingly, the NMF models evaluated using their own learned embeddings perform similarly to ChatGPT. This suggests that providing ChatGPT with iterative feedback makes it as effective as a supervised model through the lens of our evaluation pipeline. Based on these findings, we answer **RQ2** by showcasing that ChatGPT is superior to the baselines in a conversational setting.

#### 4.3 Exploring Popularity Bias in Recommendation

Our final experiment is an attempt to answer **RQ3: Does ChatGPT exhibit popularity bias in recommendation?** Due to the amount of tests performed, it became apparent that ChatGPT prefers certain recommendations over others; indicating popularity bias. Visualizations of item recommendation frequency are shown in Figure F.2 in the Appendix for further reference. With this evidence **RQ3** could be answered. However, it is more worthwhile to look for a solution to mitigate this popularity bias.

The factor *allow\_popular* indicates whether we allow ChatGPT to use popular recommendations. When *allow\_popular* = False we add the additional instruction: “Try to recommend movies that are less popular,” to all prompts. The *temperature* controls the stochasticity of the outputs. The p-values for the statistical tests indicate that all individual effects and interactions are significant for novelty, but only *allow\_popular* is significant for other metrics. TukeyHSD

for *allow\_popular* shows a significant reduction in performance when restricting the recommendation of popular items. A higher *temperature* decreases the recommendations' ILS considerably.

The main results of interest are how these factors influence novelty. A TukeyHSD comparison for novelty indicated that a high temperature and restricting popular recommendation has a profound effect on recommendation variety. If we wish to maximize novelty, we would choose to use *temperature* = 1 and *allow\_popular* = *False*. The effect of this is better seen in Figure 2 which shows a reduction of the short-tail in item frequency.

## 5 CONCLUSION

We developed an evaluation pipeline centered around ChatGPT, positioning it as an item-based, top-N conversational recommendation system. While previous studies [10, 12] have examined ChatGPT's proficiency in selecting optimal recommendations from a candidate pool, our aim was to focus on a more realistic scenario wherein the user does not provide ChatGPT with candidate items ahead of time. We found that reprompting ChatGPT with feedback in conversation has a significant impact on recommendation performance over single prompt instances. We further showed that ChatGPT significantly outperforms both random and traditional recommender systems, highlighting the utility of its robust domain knowledge in zero-shot recommendation. Finally, we examined ChatGPT's tendency towards popularity bias and proposed strategies to counteract it and produce more novel recommendations.

Future work will examine other LLM models, CF models, and other domains, as well as evaluating the collaborative information picked up by ChatGPT using our setup.

## 6 ACKNOWLEDGEMENTS

This work was partially supported by NSF OIA-2344533 and NSF DRL-2026584.

## 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. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [2] Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. 2011. Second workshop on information heterogeneity and fusion in recommender systems (HetRec2011). In *Proceedings of the fifth ACM conference on Recommender systems*. 387–388.
- [3] Yashar Deldjoo. 2024. FairEvalLLM. A Comprehensive Framework for Benchmarking Fairness in Large Language Model Recommender Systems. *arXiv preprint arXiv:2405.02219* (2024).
- [4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [5] Yunfan Gao, Tao Sheng, Youlin Xiang, Yun Xiong, Haofen Wang, and Jiawei Zhang. 2023. Chat-rec: Towards interactive and explainable llms-augmented recommender system. *arXiv preprint arXiv:2303.14524* (2023).
- [6] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.
- [7] Chengkai Huang, Tong Yu, Kaige Xie, Shuai Zhang, Lina Yao, and Julian McAuley. 2024. Foundation Models for Recommender Systems: A Survey and New Perspectives. *arXiv preprint arXiv:2402.11143* (2024).
- [8] Mathias Jesse, Christine Bauer, and Dietmar Jannach. 2022. Intra-list similarity and human diversity perceptions of recommendations: the details matter. *User Modeling and User-Adapted Interaction* (2022), 1–34.
- [9] Marius Kaminskis and Derek Bridge. 2016. Diversity, serendipity, novelty, and coverage: a survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM Transactions on Interactive Intelligent Systems (TiIS)* 7, 1 (2016), 1–42.
- [10] Wang-Cheng Kang, Jianmo Ni, Nikhil Mehta, Maheswaran Sathiamoorthy, Lichan Hong, Ed Chi, and Derek Zhiyuan Cheng. 2023. Do LLMs Understand User Preferences? Evaluating LLMs On User Rating Prediction. *arXiv preprint arXiv:2305.06474* (2023).
- [11] Bing Liu et al. 2011. *Web data mining: exploring hyperlinks, contents, and usage data*. Vol. 1. Springer.
- [12] Junling Liu, Chao Liu, Renjie Lv, Kang Zhou, and Yan Zhang. 2023. Is chatgpt a good recommender? a preliminary study. *arXiv preprint arXiv:2304.10149* (2023).
- [13] Costas Mavromatis, Petros Karypis, and George Karypis. 2024. SemPool: Simple, robust, and interpretable KG pooling for enhancing language models. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 154–166.
- [14] Sheshera Mysore, Andrew McCallum, and Hamed Zamani. 2023. Large language model augmented narrative driven recommendations. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 777–783.
- [15] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *arXiv:2203.02155 [cs.CL]*
- [16] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
- [17] 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. *arXiv:2307.14225 [cs.IR]*
- [18] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. 285–295.
- [19] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. *arXiv:1904.06690 [cs.IR]*
- [20] Cynthia A Thompson, Mehmet H Goker, and Pat Langley. 2004. A personalized system for conversational recommendations. *Journal of Artificial Intelligence Research* 21 (2004), 393–428.
- [21] Arpita Vats, Vinija Jain, Rahul Raja, and Aman Chadha. 2024. Exploring the Impact of Large Language Models on Recommender Systems: An Extensive Review. *arXiv preprint arXiv:2402.18590* (2024).
- [22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.
- [23] Xi Ye and Greg Durrett. 2022. The unreliability of explanations in few-shot in-context learning. *arXiv preprint arXiv:2205.03401* (2022).
- [24] Li Yujian and Liu Bo. 2007. A normalized Levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence* 29, 6 (2007), 1091–1095.
- [25] Hamed Zamani, Johanne R Trippas, Jeff Dalton, Filip Radlinski, et al. 2023. Conversational information seeking. *Foundations and Trends® in Information Retrieval* 17, 3–4 (2023), 244–456.
- [26] Yuhui Zhang, Hao Ding, Zeren Shui, Yifei Ma, James Zou, Anoop Deoras, and Hao Wang. 2021. Language models as recommender systems: Evaluations and limitations. (2021).



## A APPENDIX A: PARAMETER GLOSSARY

Table A.1: Parameterization of System

$P$	The number of prompts to which the recommender responds to. Including the initial prompt.
$k$	The number of recommendations to generate at the $p^{th}$ prompt.
$k_f$	The number of recommendations that we request from the recommender at the <i>final</i> prompt. These recommendations are ideally a summarized and tuned set of recommendations, after a series of feedback reprompts.
$example\_size$	Determines the number or fraction of interaction tuples $(r_{ui}, i)$ to include in the example set $\mathbb{E}_u$ for a user $u$ .
$eval\_size$	Determines the number or fraction of interaction tuples $(r_{ui}, i)$ to include in the evaluation set $\mathbb{T}_u$ for a user $u$ .
$prompt\_style$	Specifies how the initial prompt will be constructed, namely one of the options in the set: {'zero', 'one', 'CoT'}.
$q$	The $q^{th}$ quantile of pairwise similarity for an item. Specifies the strictness of weighting in relevancy matching by reserving weighting privilege to only a subset of comparable items for each item.
$title\_threshold$	Similarity threshold in which to accept a recommended title as a match in NLS.
$model$	The recommender model component of the simulation pipeline.
$temperature$	Influences the stochasticity of ChatGPT's responses.
$random\_state$	Seed used to initialize all random components of the system.
$allow\_popular$	A boolean parameter which when <i>False</i> injects the following into prompts: "Try to recommend movies that are less popular".

## B APPENDIX B: ANALYZING THE EFFECT OF EMBEDDING CONTENT

As a prerequisite to our main experiments, we first validate the content similarity approach by studying how the content used to generate the item embeddings could impact the results. The similarity between items forms the basis of how relevant recommendations are determined. Hence, it is important that their representations allow for a valid comparison. To get an idea of how the amount of content impacts similarity globally in the dataset, we examine the distributions of pairwise cosine similarity for each content level in Figure B.1.

It is evident that introducing more content makes similarity more discriminative, moving the distribution closer to the midpoint of the similarity range. However, incorporating additional information

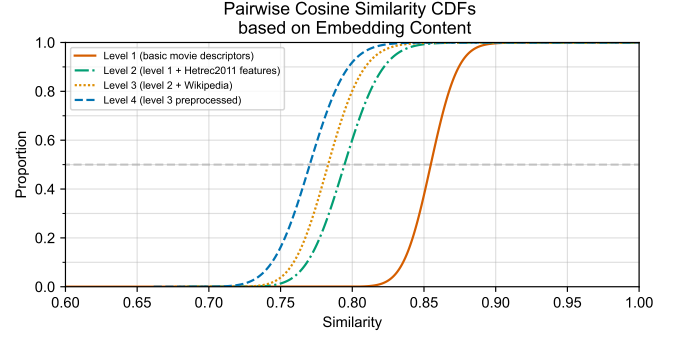


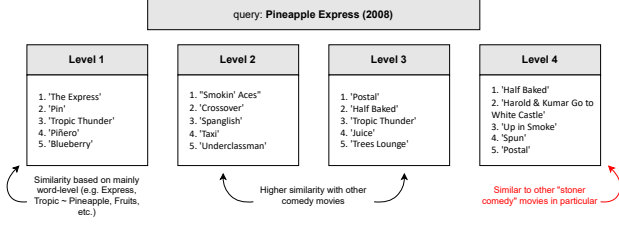
Figure B.1: CDFs of item pairwise cosine similarity. Levels are based on the amount of content contained in the sentence embeddings produced by *text-ada-embedding-002*. The level 4 content level contains the same content as level 3, excluding stop words and the top 5% most frequent word-level tokens.

Table B.1: Mean metric values for different content levels. The model used is GPT-3.5-turbo with *prompt\_style* = 'zero' for  $P = 1$  prompts. Scores are based on  $k = 20$  recommendations matched against the evaluation set. ANOVA (and Kruskal-Wallis for \* in ILS) are used to find p-values. We find that content level does have a significant effect on each metric at  $\alpha = 0.01$ .

Content	Prec.	nDCG	ILS	MAP
1	54.0	55.6	85.7	57.5
2	58.4	61.4	80.3	64.1
3	58.1	61.9	80.4	65.4
4	58.3	61.5	78.7	64.9
p-value	< 0.01	< 0.01	* < 0.01	< 0.01

appears to provide diminishing returns, as indicated by the similar distributions of levels 2-4. We test statistical significance of these content levels with respect to our metrics in table B.1. We find that for the specified tests, the level of content used in the embeddings is statistically significant for each metric at  $\alpha = 0.01$ .

A further analysis for precision, nDCG and AP using TukeyHSD revealed significantly different groupings; and further findings using Dunn's test with ILS were consistent with those shown from the initial visualization of the similarity distributions. However, it is interesting to note that even as overall pairwise similarity decreases as the content level increases, the performance also increases. This is in conflict with an initial hypothesis that a greater overall similarity between items makes the process of determining relevant matches less selective. Content levels 2-3 do not produce a significantly different effect from each other, so any of them can be safely chosen to proceed in experimentation. As stated in the main text, we choose content level 4 embeddings due to a qualitative assessment that they showcase more reasonable similar items. An example of this can be seen in Figure B.2.



**Figure B.2: Comparison of the top 5 most similar items to the movie ‘Pineapple Express (2008)’ based on the content level.**

## C APPENDIX C: METRIC FORMULATIONS

In addition to the typical metrics Precision, nDCG, and MAP, we compute several others to derive more information about the model’s behavior. The **Intralist Similarity (ILS)** metric quantifies how similar items in the recommendation list  $\hat{R}_u^{(p)}$  for user  $u$  at prompt  $p$  are to one another [8]. In this case, ILS is computed as the sum of all pairwise similarities between items  $i, j \in \hat{R}_u^{(p)}$ , divided by the total number of comparisons:

$$ILS_u^{(p)} = \frac{\sum_{i,j \in \hat{R}_u^{(p)}, i \neq j} \text{sim}(\vec{v}_i, \vec{v}_j)}{(|\hat{R}_u^{(p)}|(|\hat{R}_u^{(p)}| - 1))/2} \quad (3)$$

Where  $\text{sim}(\vec{v}_i, \vec{v}_j)$  is the cosine similarity between item embeddings  $\vec{v}_i, \vec{v}_j$ .

**Recall** measures how many items are retrievable from the entire catalog of items [9]. We modify this slightly to include the quantile threshold  $\epsilon_{q_i}$  for an item  $i$  at quantile  $q$ . This metric captures how many items of user  $u$  can be effectively matched by a recommended item  $j$  from ChatGPT at prompt  $p$ , and is computed as:

$$Recall_u^{(p)} = \frac{\sum_{i \in S_u} \sum_{j \in \hat{R}_u^{(\leq p)}} \text{match}(i, j)}{|S_u|} \quad (4)$$

$$\text{match}(i, j) = \begin{cases} 1, & \exists j \in \hat{R}_u^{(\leq p)} \text{ where } \text{sim}(\vec{v}_i, \vec{v}_j) \geq \epsilon_{q_i} \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

Where  $\hat{R}_u^{(\leq p)} = \bigcup_{n=1}^p \hat{R}_u^{(n)}$  is the set of recommendations made for user  $u$  as of current prompt  $p$ , and  $\text{sim}(\vec{v}_i, \vec{v}_j)$  is the cosine similarity between item embeddings  $\vec{v}_i, \vec{v}_j$ . The placeholder variable  $S_u := F_u$  (feedback set) or  $S_u := T_u$  (evaluation set) depending on the prompting stage. We only count unique occurrences where this is true to show that ChatGPT has approximately recommended an item.

**Novelty** is considered to be the inverse of an item’s popularity [9]:  $1 - \text{Popularity}_j$ , where we compute  $\text{Popularity}_j$  across the same pipeline configuration by collecting how many times an item  $j$  was recommended to all users  $u \in U$  across replicates  $\tau$ . Formally, we define this as:

$$\text{Popularity}_j = \frac{\sum_{u \in U} \text{occurs}(j, \mathfrak{R}_u)}{|U| \cdot \tau} \quad (6)$$

$$\text{occurs}(j, \mathfrak{R}_u) = \begin{cases} 1, & j \in \mathfrak{R}_u \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

Where  $\mathfrak{R}_u^{(p)} = \bigcup_{p=1}^P \hat{R}_u^{(p)}$  is the set of recommendations made across all prompts  $P$  for user  $u$ . User novelty is computed by summing popularity scores for each item  $j \in \mathfrak{R}_u$  and averaging by the total number of recommended items for each user:

$$\text{Novelty}_u = \frac{\sum_{j \in \mathfrak{R}_u} 1 - \text{Popularity}_j}{|\mathfrak{R}_u|} \quad (8)$$

We define a new metric the **Unmatched Ratio (UR)** to measure the fraction of unmatched recommendations made for a user  $u$  throughout a conversation. The purpose of this metric is to validate the pipeline by ensuring that other metrics are not being significantly skewed from an inability to match ChatGPT’s recommendations to a database item. This is simply defined by:

$$UR_u = \frac{|\{i \in \mathfrak{R}_u \wedge i \notin I\}|}{|\mathfrak{R}_u|} \quad (9)$$

Where  $I$  is the set of items,  $\mathfrak{R}_u$  is the set of recommendations made across all prompts.

## D APPENDIX D: BASELINE RECOMMENDER DETAILS

The NMF parameters are found using grid search with a 5% validation split taken from the non-evaluation items. Training is carried out for 15,000 updates using SGD, with model parameter restoration based on maximum RMSE for the validation set. Using this approach, we find the optimal parameter set as regularization coefficient  $\lambda = 0.05$ , step size  $\alpha = 1.2$ , and latent vector dimensionality  $d = 50$ .

**NMF-item** constructs a set of  $k_f$  recommendations for a user  $u$  by first building individual lists of  $k_f$  unique recommendations for each positive item in the example set, creating a pool  $P_u$  of possible recommendations where  $|P_u| = k_f * |E_u|$ . This pool is then reduced to the top  $k_f$  items with the highest similarity to the items in  $E_u$ . **NMF-user** produces recommendations based on the top  $k_f$  items most similar to the user in the user-embedding space. Both models use their own learned embeddings to produce recommended titles. We also vary whether relevancy matching is to be performed using the learned embeddings of NMF, or the GPT-3 embeddings. When using the GPT-3 embeddings, NMF has obviously learned a very different representation for each item because social information has been incorporated. Therefore, what may be an otherwise good recommendation when considering user overlap may not be scored highly when evaluated with content-based information only. This inclusion is made regardless to ensure a fair comparison since the only change to the pipeline is how recommendations are produced. We otherwise include the NMF’s learned embeddings for relevancy matching to give a more accurate assessment of its performance and the overall effectiveness of the proposed evaluation approach. The results of the evaluation of the NMF model on the interactions in the evaluation split  $T_u$  can be seen in Table D.1.

**Table D.1: Average NMF performance as measured on the evaluation set outside of the proposed methodology.**

RMSE	Prec.@5	Recall@5	MAP@5	nDCG@5
1.181	79.6	5.2	67.2	55.3



## E APPENDIX E: PROMPT EXAMPLES

Initial Prompt	Output by ChatGPT at $p=1$
<b>Zero-Shot</b> <p>A user liked the following movies: The Godfather (1972):like, There's Something About Mary (1998):like, ...  The user disliked the following movies: South Park: Bigger Longer &amp; Uncut (1999):dislike, Maverick (1994):dislike, ...  Based on the user's likes and dislikes, you must recommend <b>{k}</b> other movies released before <b>{release_cutoff}</b> that they may also enjoy.  Sort your recommendations by your confidence that the user will like the movie and do not provide explanations for each recommendation.</p>	<b>Zero-Shot</b> <ol style="list-style-type: none"> <li>1. The Shawshank Redemption (1994)</li> <li>2. Pulp Fiction (1994)</li> <li>3. Forrest Gump (1994)</li> <li>4. The Silence of the Lambs (1991)</li> <li>5. The Matrix (1999)</li> </ol>
<b>Few (One)-Shot</b> <p>Your task is to recommend movies that a user might enjoy based on their likes and dislikes. Here is an example:  Task: A user liked the following movies: Dummy (2002):like, Melvin Goes to Dinner (2003):like, ...  The user disliked the following movies: O Convento (1995):dislike, Men with Guns (1997):dislike, ...  Based on the user's likes and dislikes, you must recommend <b>{k}</b> other movies released before <b>{release_cutoff}</b> that they may also enjoy.  Sort your recommendations by your confidence that the user will like the movie and do not provide explanations for each recommendation.  Response:  1. Kagemusha (1980)  2. Arrowhead (1953)  ...  <b>{k}</b>: Junior (1994)  These recommendations are all sorted in descending order based on similarity to the user's preferences. Given this example, complete the following:  Task: <b>{Zero-Shot Prompt}</b>  Response:</p>	<b>Few (One)-Shot</b> <ol style="list-style-type: none"> <li>1. The French Connection (1971)</li> <li>2. The Departed (2006)</li> <li>3. The Shining (1980)</li> <li>4. The Social Network (2010)</li> <li>5. The Godfather Part II (1974)</li> </ol> <p>(Same as Zero-Shot response, with presumed additional ordering)</p>
<b>Chain-of-Thought (CoT)</b> <p>Your task is to recommend movies that a user might enjoy based on their likes and dislikes. Here is an example:  Task: A user liked the following movies: Dummy (2002):like, Melvin Goes to Dinner (2003):like, ...  The user disliked the following movies: O Convento (1995):dislike, Men with Guns (1997):dislike, ...  Based on the user's likes and dislikes, you must recommend <b>{k}</b> other movies released before <b>{release_cutoff}</b> that they may also enjoy.  Sort your recommendations by your confidence that the user will like the movie and do not provide explanations for each recommendation.  Response:  Because the user likes Jingi naki tatakai (1973) and dislikes O Convento (1995), I would recommend:  1. Kagemusha (1980)  Because the user likes Fort Apache (1948) and dislikes Breaking and Entering (2006), I would recommend:  2. Arrowhead (1953)  ...  Because the user likes Dummy (2002) and dislikes O Convento (1995), I would recommend:  <b>{k}</b>: Junior (1994)  These recommendations are all sorted in descending order based on similarity to the user's preferences. Given this example, complete the following:  Task: <b>{Zero-Shot Prompt}</b>  Response:</p>	<b>Chain-of-Thought (CoT)</b> <p>Because the user likes Star Wars: Episode V - The Empire Strikes Back (1980) and dislikes Singin' in the Rain (1952), I would recommend:  1. Star Wars: Episode VI - Return of the Jedi (1983)  Because the user likes Honey, I Shrunk the Kids (1989) and dislikes Singin' in the Rain (1952), I would recommend:  2. Honey, I Blew Up the Kid (1992)  ...  Because the user likes The Ice Storm (1997) and dislikes Singin' in the Rain (1952), I would recommend:  5. American Beauty (1999)  These recommendations are all sorted in descending order based on similarity to the user's preferences.  (ChatGPT mimics like/dislike reasoning steps in its output)</p>

Example set liked items
 Example set disliked items
 Instructional text
 Synthetic liked/disliked items
 Synthetic recommendations

Figure E.1: Initial prompt choices. Parameter injection is in bold and contained in '{-}' but is represented as only the value in the actual text.

Reprompting with Feedback
<p>The user would not enjoy the following recommendations: The Big Lebowski (1998):dislike, American Beauty (1999):dislike, ...  The user would enjoy these recommendations: Pulp Fiction (1994):like, Fight Club (1999):like, ...  Recommend <b>{k}</b> additional movies released before <b>{release_cutoff}</b> that the user may enjoy, based on what you have learned about their preferences.  Do not reuse any recommendation that you see here.  Sort your recommendations by how confident you are that the user will like a movie.  Do not provide explanations for each recommendation."</p>
Reprompting and Final Evaluation
<p>The user would not enjoy the following recommendations: American Beauty (1999):dislike.  The user would enjoy these recommendations: The Departed (2006):like, The Truman Show (1998):like, The Prestige (2006):like, ...  Based on what you have learned about the user's preferences, recommend a final list of <b>{k_f}</b> additional movies released before <b>{release_cutoff}</b> that the user might enjoy.  This list should summarize the user's preferences, and be sorted by how confident you are that the user will like a movie.  Do not provide explanations for each recommendation.</p>

Positive feedback
 Negative feedback
 Instructional text

Figure E.2: Re-prompting for incorporating feedback mid-conversation and requesting a final recommendation list. Parameter injection is in bold and contained in '{-}' but is represented as only the value in the actual text.

## F APPENDIX F: ADDITIONAL EXPERIMENTAL FIGURES

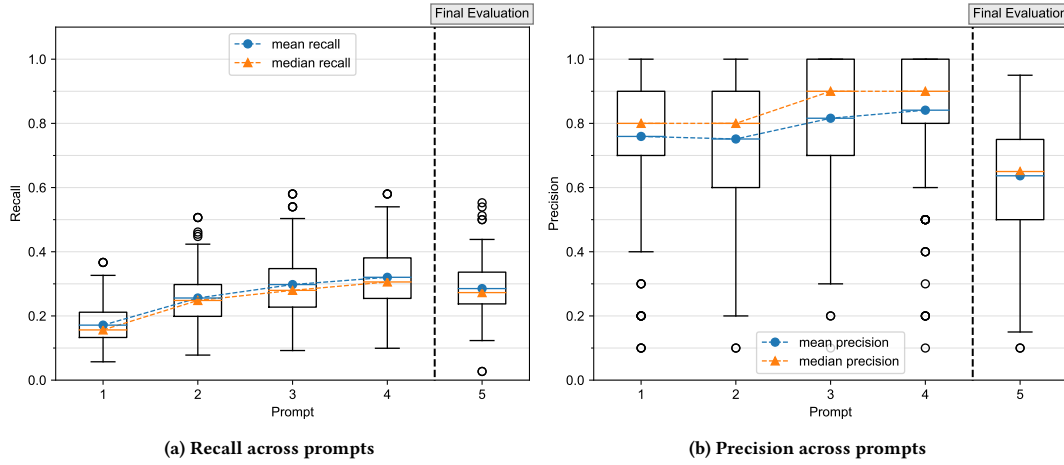


Figure F.1: Recall (a) and precision (b) distributions for different prompt numbers using best configuration from Table 1 ( $P = 5$ ,  $k = 10$ ,  $prompt\_style = 'zero'$ ). Plots show that ChatGPT continues to match unique items in the feedback set while further increasing precision.

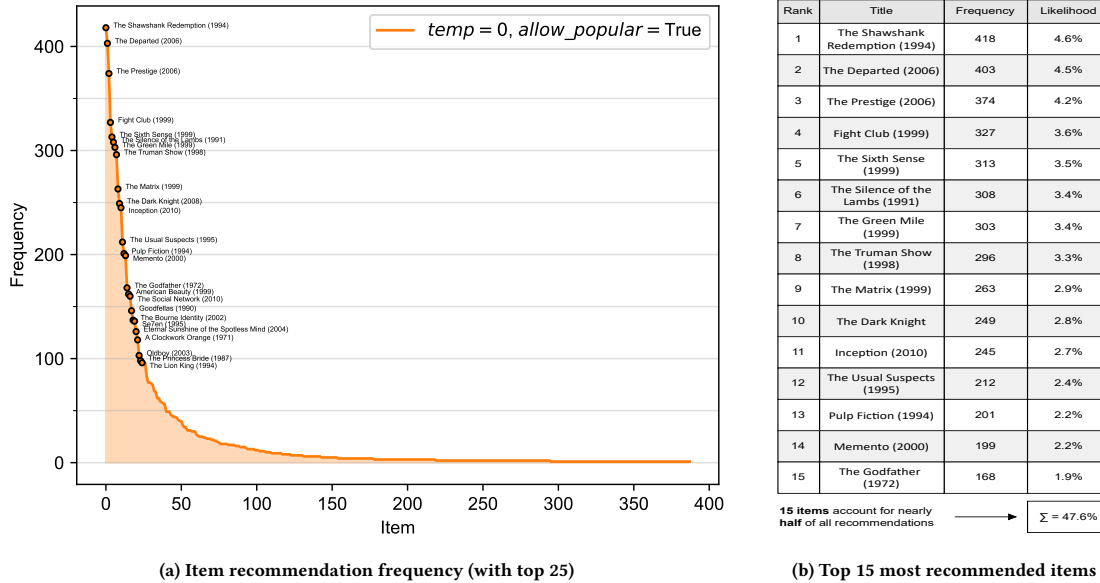


Figure F.2: Item recommendation frequency (a) and top 15 most recommended items (b) across 9,000 total recommendation instances. Model used has base parameters  $P = 5$ ,  $k = 10$ ,  $prompt\_style = 'zero'$ , and popularity parameters  $temperature = 0$ ,  $allow\_popular = True$ .