8-2023

Evaluating chatGPT for recommendation: how does the ability to converse impact recommendation?

Kyle Spurlock
University of Louisville

EVALUATING CHATGPT FOR RECOMMENDATION: HOW DOES THE ABILITY TO CONVERSE IMPACT RECOMMENDATION?

By
Kyle Spurlock
M.S. in Computer Science, 2023

A Thesis
Submitted to the Faculty of the
J.B Speed School of Engineering of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Masters of Science
in Computer Science

Department of Computer Engineering and Computer Science
University of Louisville
Louisville, Kentucky

August 2023

EVALUATING CHATGPT FOR RECOMMENDATION: HOW DOES THE ABILITY TO CONVERSE IMPACT RECOMMENDATION?

By

Kyle Spurlock
M.S. in Computer Science, 2023

Thesis approved on

August 7, 2023

by the following Thesis Committee:

Thesis Director
Dr. Olfa Nasraoui

Dr. Andrew Karem

Dr. Dan Popa

DEDICATION

I dedicate this thesis to all those that I cherish in this life: my parents and grandparents, my brother, my beloved partner, my friends, and my mentors. To all those who encouragement has helped me reach this point. I am eternally grateful for their unyielding love and support.

ACKNOWLEDGMENTS

I would first like to acknowledge my advisor and mentor Dr. Olfa Nasraoui for her unparalleled wisdom, patience, and kindness throughout work on this thesis. I truly could not have accomplished this without her words of encouragement and insight.

I would like to further extend my thanks to the members of my committee, Dr. Andrew Karem and Dr. Dan Popa for their time in reviewing my work and for providing valuable feedback and potential expansions for future work.

Lastly, I would like to acknowledge my lab mate Çağla for her help in devising the methodology used in this study.

ABSTRACT

EVALUATING CHATGPT FOR RECOMMENDATION: HOW DOES THE ABILITY TO CONVERSE IMPACT RECOMMENDATION

Kyle Spurlock

August 7, 2023

Recommendation algorithms have become an absolute necessity in the modern world to avoid information overload. However, the interaction between the human and the system is largely superficial and without any real contact. If you are given poor recommendations, you have no choice but to sift through mountains of content on your own until the model learns to accommodate your tastes more. This is bad for business as well as the consumer. Recently, large language models like ChatGPT have seen a significant rise in popularity due to their ease of use and wide range of knowledge. It has now become nearly as easy to ask ChatGPT a question as it is to search for it online yourself. Due to their domain knowledge and transferability, they have frequently been evaluated as possible tools for recommendation in recent years. However, most existing studies exploring LLM as recommendation agents focus only on a single input/output basis; neglecting one of the major benefits that LLM have expanded upon. That is, the ability to converse and respond to feedback dynamically.

In this thesis we investigate how effective ChatGPT is as a recommender in its natural use case: as a conversational, direct top-n recommendation system. We build an evaluation pipeline around ChatGPT that allows us to provide iterative feedback throughout the course of a conversation to simulate how a user would actually interact

with it. We additionally explore whether popularity bias is present in ChatGPT's recommendations, and how it compares against baseline models.

Our findings indicate that reprompting ChatGPT with feedback on its recommendations has a signficant impact on precision. Lastly, we show that popularity bias is present in ChatGPT's recommendations but can be easily mitigated through a combination of prompt engineering and raising the temperature parameter of the model.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Automated recommendation has become a prominent and ubiquitous practical application of machine learning (ML) with potential benefit to both businesses and consumers. As the amount of content available for viewing or purchasing online is ever-increasing, it has become necessary to automate the task of filtering through the vast amount of choices available to humans to avoid information overload. State of the Art recommender systems rely on ML algorithms that aim to learn from the patterns of activities and feedback of humans in order to identify and recommend a narrower set of options that are most in-line with these patterns. These patterns manifest in many interactions one has on the world wide web: as implicitly as a view or a click; or explicitly as a rating or purchase. However, this mode of interaction is largely superficial. In most recommendation algorithms implemented in practice there is no direct line of communication between the human and the model. While it can be argued that one may not want to argue with an AI about their preferences when trying to choose a movie to watch or a book to read; the possibility to do so presents an interesting opportunity for a model to learn directly from the user it serves. This is in contrast to the more passive consumption of the user's activity data when building AI recommendation models.

The task of introducing conversation into recommendation has been addressed in the past with varying degrees of success. When Natural Language Processing (NLP) was far less advanced, chatbots such as [1] were able to tailor the recommendation experience to the user, but were restricted to only a handful of predetermined templates in their responses. This makes recommendations quick and easy, but with limited scope. However, more recently, techniques that use deep learning [2] have become far more interesting and surprising to engage with. The encapsulation of Large Language Models (LLM) like ChatGPT into a non-technical, user-friendly interface has completely redefined what it means to use and interact with technology. In this work, we are interested in ChatGPT specifically because of this ease of access, with the assumption that anyone could go to its interface and request recommendations with little effort.

In this work we focus on evaluating the direct top-n recommendation potential of the large language model ChatGPT [3]. Recent work [4]–[6] has largely evaluated ChatGPT's recommendation potential on a *single* input/output basis [5], [6], often requiring the model to choose an option out of a predetermined set that best completes the task. We choose to structure our study in a way that is more representative of how someone would actually interact with the system, with an example given in Figure 1. We aim to evaluate how ChatGPT performs at recommendation in a natural setting, i.e. how can it generate pertinent information that is not readily and freely available? Additionally, we aim to explore another characteristic that LLM have expanded upon: the ability to converse and respond to feedback dynamically.

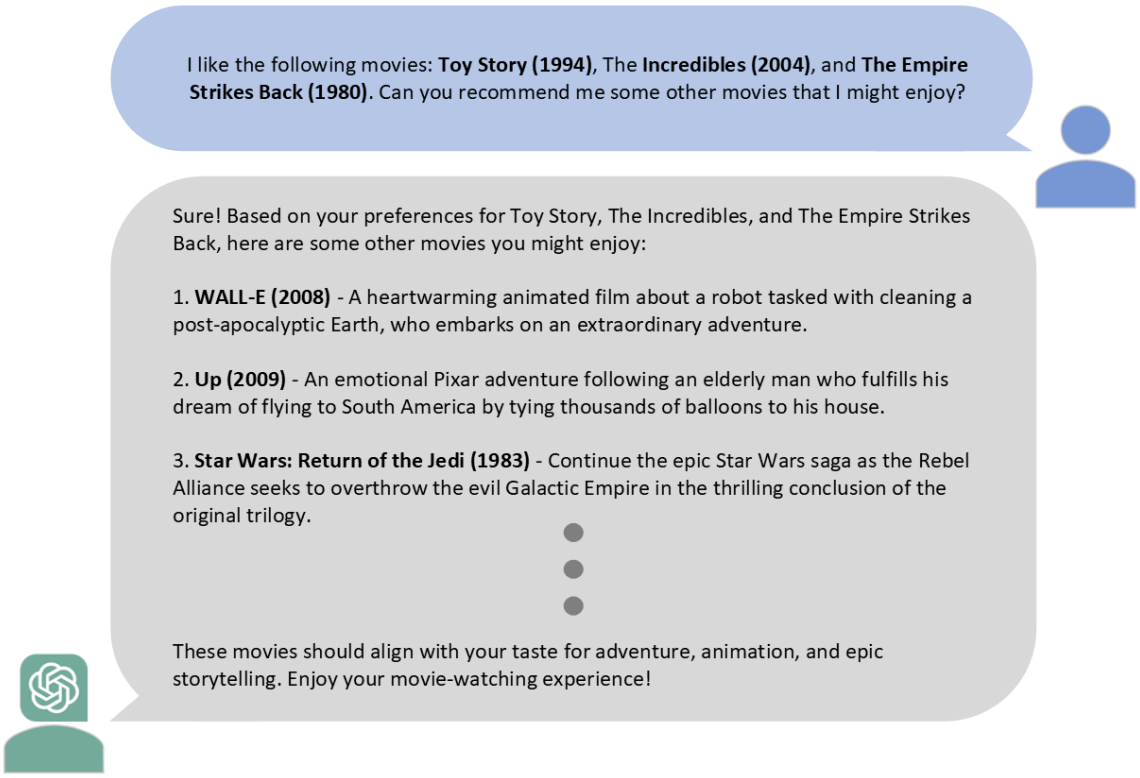


Figure 1. Motivating example for evaluating ChatGPT as a recommender.

1 Objectives

In our study, we 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 Chat-GPT exhibit popularity bias in recommendation?

CHAPTER II

LITERATURE REVIEW AND BACKGROUND

1 Deep Learning Sequence Models

Recurrent Neural Networks

Recurrent Neural Networks (RNN) are an adaptation of the traditional Feed Foward (FF) model to allow for the temporal qualities of a variable-length sequence to be captured and learned. In the most simple cases this type of model has been employed with success in a wide variety of tasks ranging from sentiment classification [7] to recommendation [8]. Effectively, a RNN unit merely compresses the information it has seen in the sequence up to time t into a hidden state h_t, and passes it along with the next element in the sequence x_{t+1} back to the cell. The hidden state at time t is computed as [8]:

$$\mathbf{h}_t = \sigma(W\mathbf{x}_t + U\mathbf{h}_{t-1}) \tag{1}$$

Where W and U are the model parameters. This recurrence repeats until the cell has processed the entire sequence from timesteps $t = 1 \dots T$, in which case it produces a final output y_T. While the generic RNN cells are effective for small sequences, they

struggle to maintain the temporal dependencies between elements as sequence length grows.

There have been two notable advancements to this architecture to further modulate the flow of information from one time step to the next, one of these being the Long Short Term Memory (LSTM) cell [9]. LSTM cells introduce several gates that oversee the process of forgetting old information and storing new information. Gated Recurrent Units (GRUs) [10] encompass this same idea, but with less parameterization and comparable performance. RNN have seen further success applied in bidirectional contexts, i.e. representations of the sequence are learned both forward and backwards using two stacked RNN cells [11].

Despite all these improvements in learning capacity, RNN variants are limited by their inability to parallelize operations across a sequence. This has disallowed them to take full advantage of the speedups afforded by new accelerator architectures like GPUs and TPUs. Additionally, while the larger parameterization of LSTM and GRU cells has allowed them to maintain a larger frame of reference surrounding a position, it is still possible to fail in capturing long-term dependencies [12].

Text Vectorization

Representing textual data in a continuous space has long been a studied problem in Information Retrieval (IR) and Natural Language Processing (NLP). In IR, a number of techniques have been used to convert text data into real-valued vectors. Allowing for better notions of similarity between documents to be derived. Among these, methods like Latent Semantic Indexing (LSI) that produce compressed latent

vectorizations of both words and documents in $\mathbb{R}^d$ have been the most successful [13].

In NLP, it became popular to utilize RNN and other deep learning models to learn embeddings for words end-to-end through gradient descent. Allowing information about how a word fits into its sentence syntactically to be encompassed into its representation. Numerous methods were borne from this that further enriched semantic and syntactic information in the embeddings [14]–[16]. This further allowed transferability of pre-trained word embeddings to other models, providing a better initialization point than starting from scratch.

Encoder-Decoder Networks

Encoder-Decoder (E-D) networks are the application of sequential networks like RNN to sequence-to-sequence tasks. E-D architectures have seen success in many different applications, with one of the most prominent being Neural Machine Translation (NMT)[10]. E-D networks are composed of an encoder model and a decoder model, and are trained on a parallel data corpus containing a source sequence $\mathbf{x}$ and a target sequence $\mathbf{y}$ [17]. The encoder processes the source sequence and compiles its hidden states h_t into a context vector c,

$$c = q(\{h_1, \ldots, h_{Tx}\}) \tag{2}$$

The decoder then takes this context vector c, and uses it alongside the target sentence $\mathbf{y}$ to model the target sequence as a joint probability of ordered conditionals [17],

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t | \{y_1, \ldots, y_{t-1}\}, c) \tag{3}$$

For NMT this effectively means that the encoder compresses one sentence in one language into c, and then the decoder uses this information along with the target sentence in another language to determine how words in the two languages correspond with one another. One of the significant issues with this is that there is only so much information that can be squashed into the context vector. Combining this with RNN's existing difficulty with handling long sequences, E-D networks often failed to achieve results comparable with statistical translation methods for some problems [17].

Attention

Attention mechanisms were introduced in E-D networks to further focus what information passed between the encoder and decoder. Bahdanau et al. (2015) [17] introduced attention by way of a seperate FF network that produces a context vector by weighting how well a position i in the source sequence and a position j in the target sequence align. Through this approach, the context vector is able to be expanded and further parameterized to weight different positions in both sequences based on how well they correspond to one another.

Luong et al. (2015) [18] introduced several simpler methods for determining how well two positions aligned in the source and target sequences. One of the most influential has been dot product attention, which computes alignment weights as,

$$a_t(s) = softmax(score(h_t, \overline{h}_s) = h_t^\top \overline{h}_s) \tag{4}$$

Where $a_t(s)$ is a variable-length alignment vector with size equal to the number of source time steps, $\overline{h}_s$ is a collection of all source hidden states, and h_t is the current target hidden state. Effectively this is used like a method of similarity between the current state in the target sequence to all states in the source. The introduction of attention significantly advanced the state of the art in NMT, and led the way towards the Transformer model.

Transformer

The Transformer introduced by Vaswani et al. [19] and shown in Figure 2; built on the E-D architecture for sequence-to-sequence prediction, but sought to eliminate many of the limitations imposed by using RNN models as the encoder and decoder. To accomplish this they dropped recurrence entirely, replacing it with a scaled variation of the dot product attention mechanism introduced by Luong et al. (2015) [18], computed as:

$$Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V \tag{5}$$

Q, K, and V stand for Query, Key, and Value respectively and d_k is the dimensionality of the keys. This naming convention is heavily inspired by IR techniques, and is analogous to a fuzzy, differentiable dictionary lookup. This attention mechanism is additionally multi-headed, meaning that they divide the dimensionality of the

inputs/ouputs amongst several 'heads' to learn how latent factors of the embeddings correlate to one another. In the self-attention layers, Q=K=V, and is representative of the encoder/decoder learning how its own sequence fits together. In the cross-attention layer, K=V are the outputs of the encoder and Q comes from the previous block of the decoder. Additionally, the encoder is bidirectional, meaning it can compute attention scores for all positions at once. The decoder, however, uses causal masking, which limits it from computing scores for positions it has not seen yet.

Since recurrence (or convolution) was previously necessary to capture temporal information within sequences, temporal information is preserved by way of positional encodings added directly to the word embeddings in the sequences. These positional encodings are computed as:

$$PE_(pos, 2i) = sin(pos/10000^{2i/d_model})$$

$$PE_(pos, 2i+1) = cos(pos/10000^{2i/d_model})$$

Where pos is the position in the sequence, i is the dimension, and d_{model} is the dimensionality used for the word embeddings and outputs of all transformations throughout the model. This allows the model to derive temporal information from the embeddings, as each position in the sequence receives its own additional frequency embedding that distinguishes it.

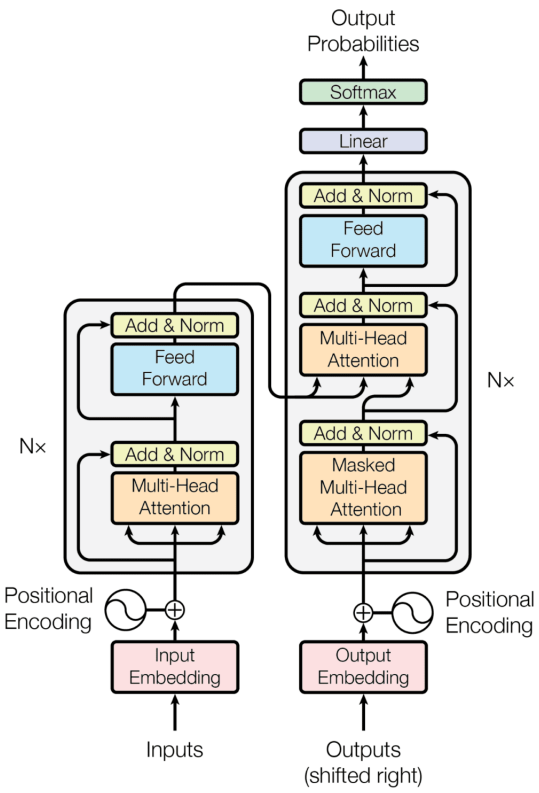


Figure 2. Transformer architecture [19]

Language Models

Because the Transformer afforded significant speed-ups to previous RNN-based architectures, it now became practical to train models on a scale never seen before. Language modeling in Transformer models was first explored by GPT-1 [12], which used only the decoder of the Transformer. Their training process first involved unsupervised pre-training on a corpus of tokens $\mathbb{U}$ for a standard language modeling objective,

$$L_1(\mathbb{U}) = \sum_i logP(u_i|u_{i-k}, \ldots, u_{i-1}; \Theta) \tag{6}$$

Which is designed merely to maximize the probability of a token u_i based on a

context window surrounding it, where k is the size of the window and Θ are the model parameters. This is followed by supervised fine-tuning for specific tasks.

After this was found to be effective, pre-trained language models quickly turned into large language models (LLM) as parameterization skyrocketed, consuming near unfathomable amounts of data and advancing the state-of-the-art across various NLP tasks when fine-tuned; and showcasing impressive transferability even without tuning for a specific task [20][21].

One of the more recent advances in LLM has been seen in gpt-3.5-turbo, the model behind ChatGPT. ChatGPT has been further optimized for conversation through Reinforcement Learning with Human Feedback (RLHF) to follow a user's requests more closely and provide a more enjoyable interactive experience [3].

Prompt Engineering

Prompt engineering is a new domain that has arisen as a result of recent LLM's in-context reasoning capabilities without fine-tuning. This burgeoning body of work aims to explore how best to communicate with LLM when asking them to perform a task in-context. Generally, there have been two predominant means of communication with LLM through prompting [21]:

- **Zero-shot**: the model is provided with only instructions and asked to complete a task.
- **Few-shot**: 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.

- **Chain-of-Thought prompting (CoT)**: 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 devised approaches are based around these archetypes and either vary the amount of information or present the task in a different way.

2 Recommender Systems

Consumers today are perpetually inundated with a never ending stream of information and content; far too much for one person to ever sort through manually. Recommender systems have emerged as the solution to this; with the general goal of recommendation being to filter immense collections of items down to a small subset that an individual may be interested in. The methods used to perform this filtering are commonly divided into two primary categories: content-based, and collaborative-based.

Content-Based

Content-filtering based recommenders work on a content level of the items to be recommended. They consider only a user's demonstrated preferences towards recommendable items and use this information to recommend similar items to the ones

the user has already positively responded to. To accomplish this, a content-based recommender is typically composed of several components [23]:

- Content Analyzer: the extraction of relevant information from non-structured data about the items to be recommended, e.g. text.
- Profile Learner: the module that collects and maintains data on user preferences to be used for future recommendations.
- Filtering Component: the module that utilizes the constructed user profile to produce new recommendations.

Collaborative-Filtering

Collaborative-filtering based recommenders utilize social information to make recommendations. Information about item preferences is learned from *all* the users, rather than from the content of the item itself. In this way, if user A and user B are found to overlap in their preferences, then what user A enjoys can be recommended to user B with confidence that they will also enjoy that item based on their similar behavior [24].

Neighbor style k-nearest neighbor style similarity methods (among the users or among the items, based on the rating data) have been the traditional approach [24]. However, more recently, Matrix factorization (MF) has become the more popular, state of the art method used in collaborative filtering. MF is a technique that is able to capitalize on user behavior by decomposing the user-item feedback matrix $A \in \mathbb{R}^{m \times n}$ into two lower rank matrices $U \in \mathbb{R}^{m \times d}$ and $V \in n \times d$ that place users i

and items j in the same embedding space with d latent factors [25]. The user U and item V matrices are learned in such a way that their product roughly approximates the feedback matrix:

$$A \approx UV^\top \tag{7}$$

This means that the dot product between a user embedding u_i and item embedding v_j can produce new ratings that did not exist prior. There are a variety of methods used to learn these user-item embeddings, like Singular Value Decomposition (SVD) or gradient descent. The latter is usually preferred to avoid overfitting, and can also incorporate other terms like L2 regularization. MF can be trained with gradient descent by minimizing the following objective function [25]:

$$J = \min \sum_{(i,j)\in A} (a_{ij} - u_i v_j^\top)^2 + \lambda(||u_i||^2 + ||v_j||^2) \tag{8}$$

Where λ controls the strength of L2 regularization on the embeddings. When the factors are enforced to be positive, MF becomes Non-negative MF or NMF [26].

3 Language Models as Recommenders

Due to the extensive domain knowledge of LLMs, they have frequently been explored for the task of recommendation in recent years. However, as expressed in the introduction, this has largely focused on tasks involving a single inputs and ouputs.

Sun et al. (2019) fine-tune the encoder-only language model BERT for sequential recommendation and achieve significant improvements over previous baselines using

RNN.

Zhang et al. (2021) [27] were among the first to explore in-context sequential recommendation with LLM, showcasing the performance of the encoder-only model [28] and GPT-2[20] across a variety of zero shot prompts. Kang et al. (2023) [6] follow this up with more recent models, with findings that show the benefit of using LLM for sequential recommendation with fine-tuning.

Liu et al. (2023) [13] evaluated ChatGPT on a number of different recommendation tasks, including: rating prediction, sequential recommendation, and direct recommendation. They find several results that outperform traditional recommendation approaches.

Gao et al. (2023) [4] further evaluate GPT-3 models as augmented language models to interface with existing recommendation systems. They further provide a number of case studies that showcase the potential of ChatGPT for recommendation.

4 Chapter Summary

In this chapter we have briefly explored the path that has lead to the development of LLM, as well as their successes over RNN in several fields like NMT and recommendation. We have further explored basic recommender systems, and discussed how prompting has influenced the development of new recommendation systems utilizing LLM with and without fine-tuning.

CHAPTER III

METHODOLOGY

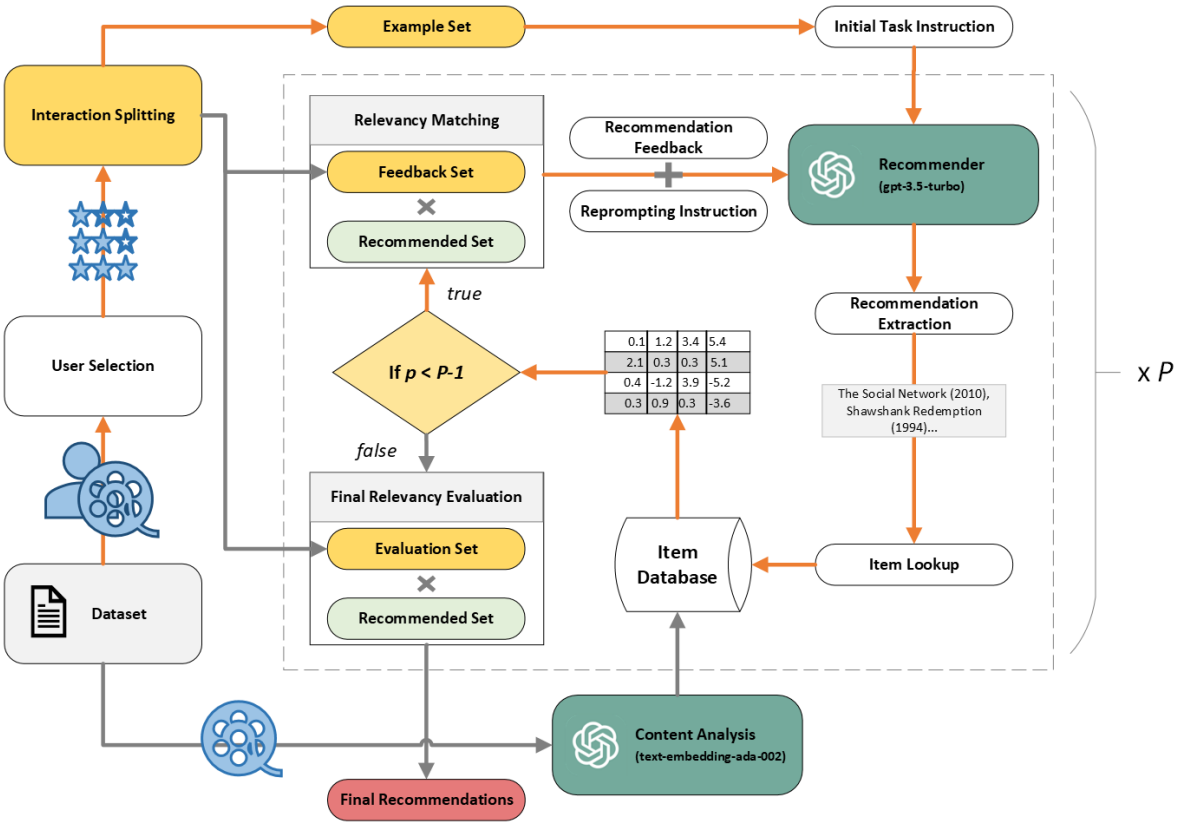


Figure 3. Proposed methodology. P=number of prompts, p=prompt number.

1 Overview of the Methodology

This section details the methodology used in this study to construct an evaluation pipeline around ChatGPT that utilizes iterative feedback. This pipeline is visualized in Figure 3. We liken the process to treating ChatGPT as a content-based

recommender. Representations or embeddings are first computed for the items to be recommended; a user profile is then learned in-context based on feedback; and future recommendations are made based on this profile. Since one major goal of a recommender system is to filter from a large quantity of content into a small, high-relevancy subset, ChatGPT is asked to finalize its recommendations at the very end of the pipeline. Therefore, the re-prompting stage is intended to explore a user's interaction space, and the final recommendation stage is intended to exploit the information learned throughout the conversation.

2 Data

Our study is built around using the HetRec 2011 dataset [29] as the ground truth for evaluation. Hetrec 2011 is an extended version of the MovieLens10M [30] dataset containing additional film information sourced from IMDB [31] and Rotten Tomatoes [32]. The added attributes that of greatest interest to our work are compared against MovieLens10M in Table 1.

MovieLens10M	HetRec 2011
Title	+Directors
Year	+Actors
Genres	+Filming locations
User-generated tags	+Country of origin

Table 1. Comparison of MovieLens10M base film attributes with those addeded by HetRec 2011

Because we seek to mimic as much of the information ChatGPT knows about these movies as possible in order to properly evaluate its recommendations, we use this additional information to construct a richer representation embedding for each

title in the content analysis stage. This is further supplemented with information crawled from Wikipedia, in an amount equal to all paragraphs between the first three headings for each article. Although Wikipedia lacks uniformity across all of its articles, this is enough to capture some general information about a movie, as well as its plot. Since some titles are based on books and may have differing articles, we attempt to grab the film-specific version of all articles by querying only pages containing an HTML template used for films (Template:Infobox_film). Of the 10,197 movies contained in Hetrec, we were able to obtain the Wikipedia articles for 9,722 movies as of 8/1/2023.

3 Content Analysis

Because we wish to evaluate ChatGPT based on its natural output and not by providing it a set of correct answers ahead of time, we need to be able to quantitatively assess the output of the model. This is made more difficult since ChatGPT is a closed system that produces only text, and not continuous representations that would allow for better measures of similarity to be derived. To get around this, the embedding model text-ada-embedding-002 provided by OpenAI is used to generate these continuous representations for the movies in Hetrec. Since both ChatGPT and the embedding model are built on GPT-3 and use the same tokenizer, we believe that this is the best available option for capturing the information ChatGPT has learned about an item outside of having direct access to the model.

However, this idea comes with several other issues. LLM have been trained on enormous swathes of data, and it would be a difficult task to determine what exact

information a model has consumed and learned about a specific item. It could be the case that a movie appears in the training data alongside a large quantity of additional information, including: its plot, directors, information from users who like this movie, etc. In this case where extra information co-occurs with the most basic descriptors of a movie; the title, release year, and genres should be enough to retrieve a relatively robust representation for the item. In the contrary case where a movie has not appeared in the training data, an embedding would need to be constructed from scratch based on additional information. As a precaution to this issue, we generate sentence embeddings for each item in Hetrec based on increasing levels of content, shown in Figure 4.

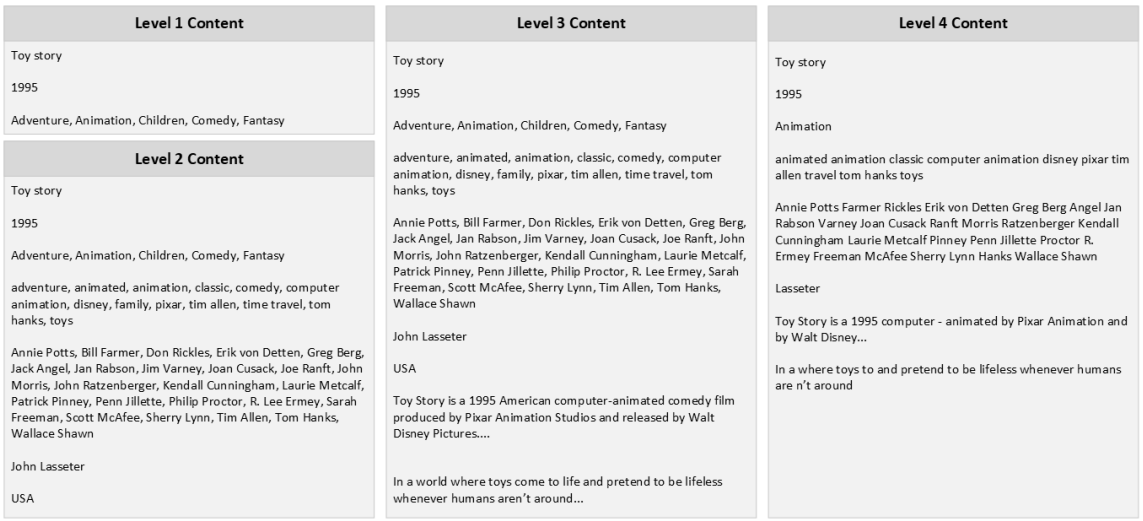


Figure 4. Different amounts of content used to learn the embeddings for the movie 'Toy Story (1995).'

Level 1 content contains only the most basic information about a movie, which would be present in smaller datasets like MovieLens100k [33]. Level 2 incorporates the extra movie attributes from Hetrec displayed in Table 1. Level 3 contains the 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. We produce this last level based on the traditional IR conception that frequently occurring words are not meaningful for similarity [13].

The embedding model produces pre-normalized sentence embeddings based on the content with dimensionality $d = 1536$ for each item. It is not disclosed how exactly these embeddings are generated, but a good guess may come from added feedfoward or attention layers to reduce a sequence $\mathbf{x} \in \mathbb{R}^{T \times d}$ to $\mathbf{y} \in \mathbb{R}^{d}$, as has been explored in other works [34], [35].

Because the embeddings come pre-normalized, our only applicable method of computing similarity is cosine similarity. We examine qualitatively how each of these different content levels affect the top-5 most similar items for a given item. This is shown by an example for the movie 'Pineapple Express (2008) in Figure 5. Based on the description that this movie is a "stoner action comedy film," similar items determined with content level 4 embeddings seem much more appropriate than those found by content level 1 embeddings.

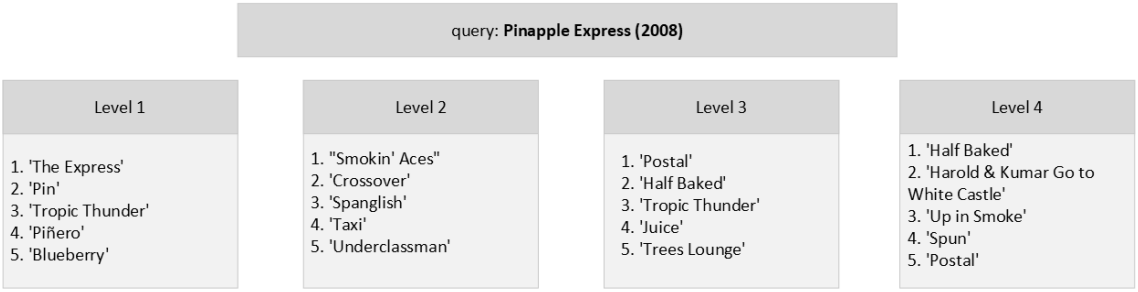


Figure 5. Comparison of the top 5 most similar items to the movie 'Pineapple Express (2008)' based on content level.

4 User & Item Selection

50 users were selected between the 50th and 75th percentile of items rated, corresponding to at least 122 total interactions. Additionally, we required that each user has provided negative reviews for at least 30 items. As long as a user satisfies these constraints, the specific choice of user is arbitrary. It is only necessary that they have had enough interactions to allow for a good estimate of their preferences. The pool of users is kept small because the output of ChatGPT is inherently nondeterministic even when randomness is minimized by setting the *temperature* parameter to the lowest value. To try to increase reproducibility and reduce variance in the reported metrics, we instead choose to perform replicate runs for each user to develop a better estimate of the model's performance.

User interactions are split into three random subsets which each serve different purposes over the course of the evaluation. We refer to these as the *example set* $\mathbb{E}_u$, *feedback set* $\mathbb{F}_u$, and *evaluation set* $\mathbb{T}_u$ for a user $u \in U$. This applies to both positively rated items taken at the typical rating threshold of 3, as well as negatively rated items with a rating lower than 3; for ratings between 1 and 5. The items in the example set are used in the initial prompt construction to provide ChatGPT with information on user's preferences. The feedback set is used solely during the *reprompting* component to help further develop ChatGPT's understanding of the user profile over several iterations. Lastly, the evaluation set 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. Otherwise, it is probable that the model could merely regurgitate those items given as examples to achieve high performance.

Splits are stratified when possible, ensuring that both the positive and negative rating distribution is similar amongst each of the subsets. This becomes important when trying to determine the relevancy of a recommended item, as ideally there will be enough positive and negative interactions from the user to estimate how they will feel about an item ChatGPT has recommended.

5 Initial Prompt Construction

The initial prompt that ChatGPT receives should inform it of its task and optionally provide additional information that helps it complete this task favorably. For our purposes we wish to paint an initial picture of a user's tastes and allow ChatGPT to build recommendations based on this. We experiment with three different prompting strategies, as may be seen in Figure 6.

Items from the example set $\mathbb{E}_u$ are injected into each prompt, and postpended with an identifier that indicates whether an item was liked or not. The example items are followed by instructional text that request specific behavior from the model, such as ranking by confidence. Other parameters are injected into the prompt that indicate the requested number of recommendations k, as well a constraint on the recommendation space by $release\_cutoff$. We specify the latter to avoid making recommendations for items not contained in our dataset. For Hetrec the most recent movie was released in 2011.

We recognize that zero-shot is the most likely style that would be used by some-

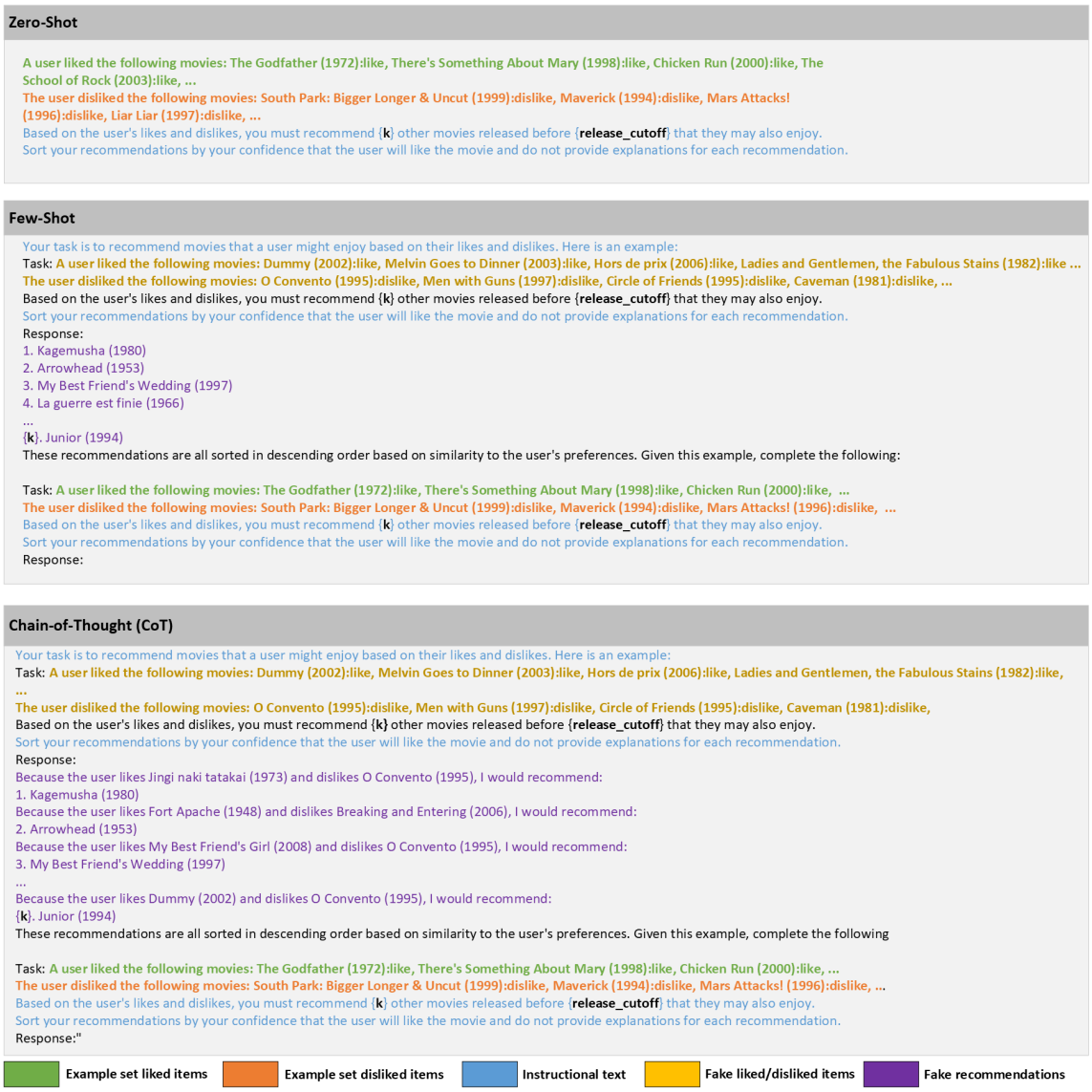


Figure 6. Initial prompt choices. Parameter injection is in bold and contained in '{-}' but is represented as only the value in the actual text.

one interacting with the system normally. For the sake of comparison we include prompting styles with examples of the task in few-shot and Chain-of-Thought (CoT) [22] format based on findings that techniques displaying higher reasoning have been widely found to increase the model's reasoning ability in turn [5], [6], [21]. The CoT format is also motivated by the work of [36] that shows a marginal increase in performance when including additional explanations. To generate fake examples for

few-shot and CoT prompts, we randomly sample $|E_u|$ liked and disliked items from all possible items and then construct k recommendations based on the similarity of other items in the data set. The reasoning displayed in CoT also incorporates ranking of results based on sorted collective similarity to the fake liked/disliked items, and aims to demonstrate to ChatGPT how it should approach the problem of making recommendations and sorting them based on confidence. We further include some smaller details that have shown to be impactful, like using '\n' to separate reasoning steps [37].

6 Recommendation, Extraction, & Mapping

Once ChatGPT has produced its completion at a given prompting stage, its recommendations must be parsed from the textual output. Because this output is natural language, some minor errors in formatting, spelling, and grammar are to be expected. In other studies [4], [5], this has been addressed as a possible issue when extracting and evaluating the recommendations. We have found little issue with formatting, as ChatGPT will generally always provide its output as a numbered list of items, similarly to what is shown in Figure 1. Explanations for each item also appear to be automatically provided when the requested number of recommendations is low ($k \leq 10$), and can be turned on or off in the initial instruction prompt. Because formatting was shown consistent with numbered lists, it is possible to extract each title using regular expressions and strip extraneous information, such as the list number and explanation, to acquire the recommended title.

After extracting the titles of each item, they are matched to an embedding that has

been learned and stored in an embedding database. As noted in [4], ChatGPT does not follow the typical ordering used for film titles, e.g. "{title}, {article} ({year})." To facilitate better matching, all item titles in Hetrec are preemptively re-ordered to match ChatGPT's natural output format of "{article} {title} ({year})." Furthermore, it is highly likely that a recommended title does not exactly match the title stored in the database. For example, a recommendation of "Seven Samurai (1953)" would fail to match with the actual title of "Seven Samurai (1954)" due to a single error in the release year. To avoid this problem, the embeddings for the recommended titles are retrieved from the database using a fuzzy lookup by Normalized Levenshtein Similarity (NLS)[38]:

$$NLS(X,Y) = 1 - \frac{2 \cdot GLD(X,Y)}{\alpha \cdot (|X| + |Y|) + GLD(X,Y)} \tag{9}$$

With,

$$\alpha = \max\{\gamma(a \to \lambda), \gamma(\lambda \to b), a, b, \in \Sigma\} \tag{10}$$

For an alphabet Σ and where $GLD(X,Y)$ is the Generalized Levenshtein Distance between two strings X and Y given by:

$$GLD(X,Y) = \min\{\gamma(T_{X,Y})\} \tag{11}$$

In which $T_{X,Y} = T_1 T_2 \dots T_l$ is a sequence of edit operations and $\gamma(T_{X,Y}) = \sum_{i=1}^{l} \gamma(T_i)$ is a weighting function for each edit operation. Levenshtein distance

allows for three operations, insertions: $\gamma(\lambda \to a)$; deletions: $\gamma(a \to \lambda)$; and substitutions: $\gamma(a \to b)$, which are used to transform X into Y. Typically, weights for all three operations are 1. Normally, the output of GLD is merely the number of insertions, deletions, and substitutions required to transform a string into another. NLS, however, constrains this to the range of [0, 1] and weights the number of changes proportionally to the length of the strings. If we find that a title X is similar enough to a title Y by NLS in the database, we proceed to match X with Y and return its embedding. The similarity threshold is controlled by the parameter *title_threshold*.

For out-of-dataset items that cannot be matched either exactly or with NLS, we merely exclude these from the metric computation and feedback process. This neither penalizes nor rewards the model for the error. However, as this could undoubtedly skew the evaluation process in an unpredictable way, we attempt to mitigate the amount of failed matches as much as possible by keeping a short list of the most common unmatched titles. This list is then included in the database complete with embeddings generated in the same way as was done in the content analysis step described in section 3. As revealed by the case-study in [4], it would be possible to discover embeddings for these items on the fly given their title, which has the benefit of avoiding the possibility of matching errors altogether. However, since it is debatable how much content is actually required to form a good quality embedding, we choose to treat all collected unmatched titles as if they were entries originally present in the data. This means there is a need for also collecting a similar amount of information for them as is available in Hetrec 2011. From the IMDb non-commercial datasets [31] we collect the correct release year and genre of the movie, then supplement this with

portions of its Wikipedia article as described previously. We collect 363 additional titles that are frequently mismatched.

7 Relevancy Matching

The most important part of the simulation pipeline is to estimate whether a given user would respond positively to a recommended item. For a set of recommended items R_k for user u, we estimate the rating $\hat{r}_{ui}$ for an item $i \in R_k$ by computing a weighted sum of ratings r_{uj} for $j \in \mathbb{S}_u$ as [24]:

$$\hat{r}_{ui} = \frac{\sum\limits_{j \in \mathbb{F}_u} \text{sim}(i,j) \cdot r_{uj} \cdot signf(i,j)}{\sum\limits_{j \in \mathbb{F}_u} \text{sim}(i,j) \cdot signf(i,j)} \tag{12}$$

$$sim(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{d} u_i \cdot v_i}{\sqrt{\sum_{i=1}^{d} u_i^2} \sqrt{\sum_{i=1}^{d} v_i^2}} \tag{13}$$

$$signf(i,j) \begin{cases} 1 & sim(i,j) \geq z_{>q_j} \\ 0 & sim(i,j) < z_{>q_j} \end{cases} \tag{14}$$

Where $sim(\mathbf{u}, \mathbf{v})$ is cosine similarity, and $z_{>q_j}$ is the q^{th} quantile of pairwise cosine similarities between item j and all other items in the dataset. If $\hat{r}_{uj} \geq 3$, we accept the recommendation i as a relevant. $\mathbb{S}_u := \mathbb{F}_u$ or $\mathbb{S}_u := \mathbb{T}_u$ depending on the prompting stage.

This by-item similarity threshold is put in place from a practical standpoint of considering there to be a finite number of items that are realistically comparable to

item j. For example, specifying that an item must exceed the 99^{th} quantile of an item's similarities has the practical implication of restricting item interaction to only the top 1% percent of interactions. For Hetrec, 1% of interactions for 10,197 movies means that we consider there to be 101 items that are feasibly comparable to an item of interest. This addition is made because the item representations produced by the text-ada-embedding-002 model all have relatively high cosine similarity even as more content is included for the item, making it difficult to define a threshold for item comparison by standard convention, like 0.5 for cosine similarity.

8 Reprompting with Feedback

The reprompting stage involves performing relevancy matching against recommended items, and merely informing ChatGPT which of these recommendations were good or not. 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 some extra context is added to the next prompt. Both of these prompts can be seen in Figure 7.

9 Evaluation Metrics

We base our true comparison of different methods only on the last set of k_f recommendations generated for comparison against the evaluation set $\mathbb{T}_u$. Since the entire purpose of a recommendation system is to provide a succinct set of relevant items, it would otherwise be impractical to take all $k \cdot (p-1) + k_f$ recommendations gen-

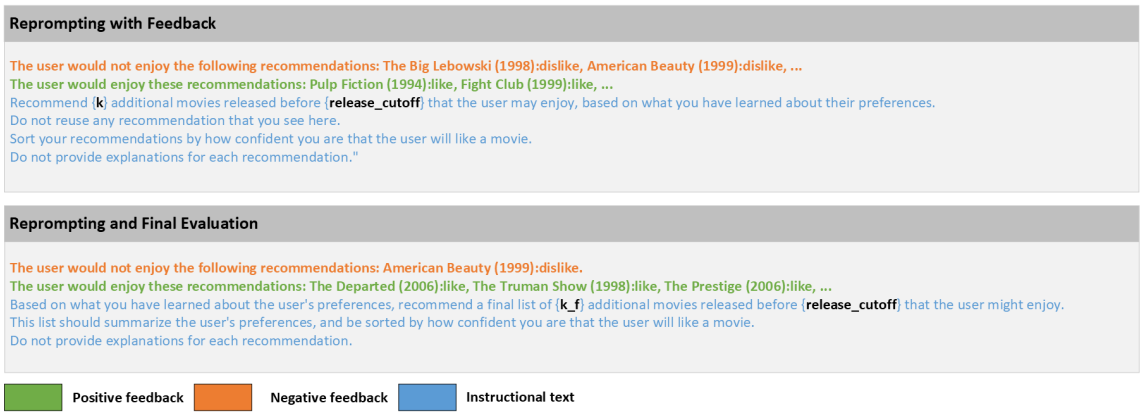


Figure 7. Re-prompting for incorporating feedback mid-conversation and requesting a final recommendation list. Parameter injection is in bold and contained in '{-} but is represented as only the value in the actual text.'

erated throughout the conversation with p prompts and use this to determine the performance of ChatGPT. The goal is to help ChatGPT explore the user's interests further, and use this to develop a more robust set of recommendations at the very end. We compute the following metrics to evaluate final performance.

Mean Average Precision

MAP is a well-known metric for evaluating recommender systems that takes into account the ranking of recommended items, as well as precision at different top-N sets [39].

First the general formula for precision in recommendation for a set of recommended items R:

$$P = \frac{\text{Number of recommendations that are relevant}}{\text{Total number of recommendations}} \tag{15}$$

Hence for a recommended list of i ranked items, the precision at the i^{th} rank would

be $P(i)$= number of relevant items up to rank i.

Average precision is then computed as:

$$AP = \frac{1}{|R|} \sum_{k=1}^{|R|} P(k) rel(k) \tag{16}$$

Where $rel(k)$ is a binary indicator that returns 1 if the i^{th} item is relevant and 0 otherwise.

The Mean Average Precision (MAP) is finally calculated based on the Average Precision for each user u

$$MAP = \frac{1}{|U|} \sum_{u \in U} AP(u) \tag{17}$$

Normalized Discounted Cumulative Gain

The normalized metric is similar to MAP since it accounts for ranking, but is more sensitive to the rank of the item. Relevant items closer to the front of the list are worth more than those placed at the end [40].

With R as the recommended list and $rel_i \in \{0, 1\}$; indicating whether the i^{th} item in the list is relevant, the Discounted Cumulative Gain (DCG) is computed as:

$$DCG = \sum_{i=1}^{|R|} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{18}$$

and

$$nDCG = \frac{DCG}{IDCG} \tag{19}$$

Where IDCG is the ideal DCG defined by the best possible ranking determined

by the set of maximally relevant items R^*:

$$IDCG = \sum_{i=1}^{|R^*|} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{20}$$

Intra-list Similarity

The ILS metric is a simple metric that provides information about how similar items in the recommendation list R are to one another [41]. It is commonly computed as the the sum of all pairwise similarities between items $i, j \in R$ divided by the total number of comparisons:

$$ILS(R) = \frac{\sum_{i \in R} \sum_{j \in R \, i \neq j} sim(i,j)}{(|R|(|R|-1))/2} \tag{21}$$

Coverage

Coverage is a metric that measures how many items are retrievable from the entire catalog of items [42]. Typically, this is synonymous with Recall, but we modify this slightly based on the quantile threshold $z_{>i}$ for an item i and user u. With this metric we would like to know how many items can be effectively 'matched' by a recommendation from ChatGPT. This is computed as:

$$Coverage = \frac{\sum_{i \in R^P \; j \in F_u} match(i,j)}{|F_u|} \tag{22}$$

Where R^P is all recommendations made across all prompts, and $match(i,j)$ is 1 when there exists some item $i \in R^P$ such that $sim(i,j) \geq z_{>j}$, and 0 otherwise.

We only count individual occurences where this is true to show that ChatGPT has approximately recommended an item.

Novelty

Novelty is considered to be the inverse of an item's popularity [42]: 1 - $Popularity(i)$, where we compute $Popularity(i)$ across the same pipeline configuration by collecting how many times an item i was recommended to all users U across replicates τ. Formally, we define this as:

$$Popularity(i) = \frac{\sum_{u \in U} occurs(i, R_u^P)}{|U| \cdot \tau} \tag{23}$$

Where $occurs(i, R_u^P)$ is an indicator function that returns 1 if $i \in R_u^P$ for a user u and 0 otherwise. Then we compute the user novelty by summing these popularity scores for each item $i \in R_u^P$ and averaging by the total number of recommended items for each user:

$$Novelty_u = \frac{\sum_{i \in R_u^p} 1 - Popularity(i)}{k \cdot (p-1) + k_f} \tag{24}$$

Unmatched Ratio

We define a new metric to measure the amount of unmatchable recommendations made for a user u, simply defined by:

$$UR_u = \frac{\{|i \in R_u^P \wedge i \notin I\}|}{k \cdot (p-1) + k_f} \tag{25}$$

Where I is the set of items in our database, R_u^P is the set of recommendations made across all prompts, k is the number of recommendations made per initial/reprompt, p is the number of prompts, and k_f is the number of recommendations made at the evaluation prompt.

10 Comparison With Other Methods

To achieve a meaningful comparison with the baseline recommendation methods, the recommender component of the tested pipeline is replaced with an alternative model to ChatGPT. This accounts for the unorthodox method of evaluation to ideally establish lower and upper bounds on performance. Namely, two additional approaches have been explored:

- Randomized
- Non-negative Matrix Factorization (NMF)

The randomized baseline is a simple model that merely builds a final list of k_f number of recommendations at random from the 10,197 items available in Hetrec.

For both options, we try to simulate how direct recommendation occurs through ChatGPT in this context, and so these additional methods are subject to the same extraction, mapping, and relevancy matching procedures even though they are not necessary.

11 Chapter Summary

In this chapter we have laid out the process of our evaluation. Detailing how we have chosen to interact with ChatGPT as a recommender, and how we plan to perform an assessment of its performance.

We further note the following parameters of our proposed 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 *final* 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', 'few', '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 shared by all random components of the system.

CHAPTER IV

EXPERIMENTAL RESULTS & ANALYSIS

1 Experimental Design

For our experiments we utilize a Randomized Complete Block Design (RCBD) [43] 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 (IV) of interest. Randomization is inherent both within the block and across blocks due to using a separate asynchronous thread to make calls to the OpenAI API for each user. Each of these threads may take approximately 200-1800 seconds to run, and parameterization is shuffled in-between. Unless otherwise specified, we use an alpha level of $\alpha = 0.05$ to determine statistical significance. We run separate ANOVAs with each metric as the DV to determine significantly different means; provided parametric assumptions hold. If the data violates normality, then a Kruskal-Wallis test is used to determine significantly different medians within groups. Post-hoc tests for significant factors are performed with TukeyHSD after ANOVA, and Dunn's multiple comparison test after Kruskal-Wallis [44].

We hold the following parameters of the system constant throughout all experiments:

- $k_f = 20$
- $example\_size = 10$
- $eval\_size = 0.33$
- $title\_threshold = 0.75$
- $q = 0.99$
- $temperature = 0$
- $random\_state = 22222$

The *temperature* parameter which increases the stochasticity of ChatGPT's responses is held constant for our main results to maximize reproducibility. However, we do vary it for one experiment, which may make the results non-reproducible.

2 Analyzing the Effect of Embedding Content

In this section we want to determine how the content used to generate item embeddings could impact the results. Since similarity between items forms the basis of how relevant recommendations are determined, it is important that their representations allow for a valid comparison. To get an idea of how the amount of content impacts global similarity, we examine the distributions of pairwise cosine similarity for each content level in Figure 8.

It is evident that introducing more content makes similarity more discriminative; however, we seem to approach a limit when incorporating additional information, as indicated by the distribution of levels 2-3.

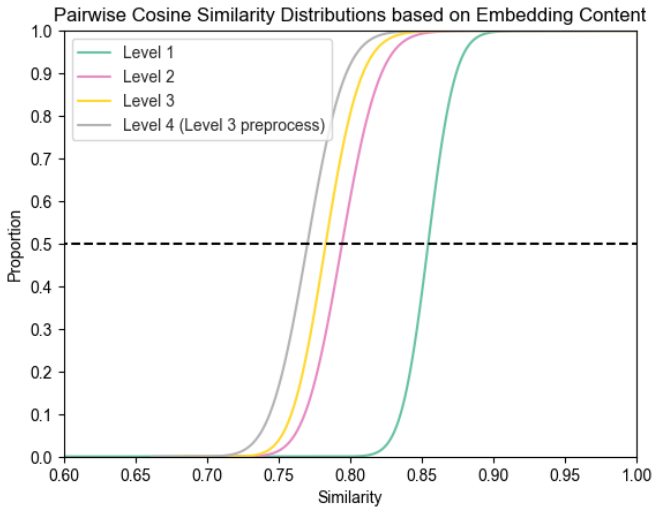


Figure 8. Cumulative distribution functions of item pairwise cosine similarity. Levels are based on the amount of content contained in the sentence embeddings produced by text-davinci-002, shown in Figure 4. The level 4 content level contains the same content as level 3, but with stop words and and the top 5% most common word-level tokens removed before embedding.

To determine whether these visible differences are actually statistically significant we perform an ANOVA test. In order to use a parametric test like ANOVA, the data must be approximately normal and identically distributed with homogeneity of variance (NID)[45], which we verify by plotting residuals in Figure 9. The plots for the residuals of ILS indicate that an ANOVA is not appropriate there is an extreme violation of normality, so we instead utilize the Kruskal-Wallis non-parametric test to determine if there is a significant difference in the levels. P-values for these tests are given along with the mean metric values for each level in Table 2. We find that for the specified tests, the level of content used in the embeddings is statistically significant at $\alpha = 0.01$.

A further analysis for precision, nDCG and AP using TukeyHSD to determine significantly different groupings is shown in Figure 10(a). The Dunn's test results

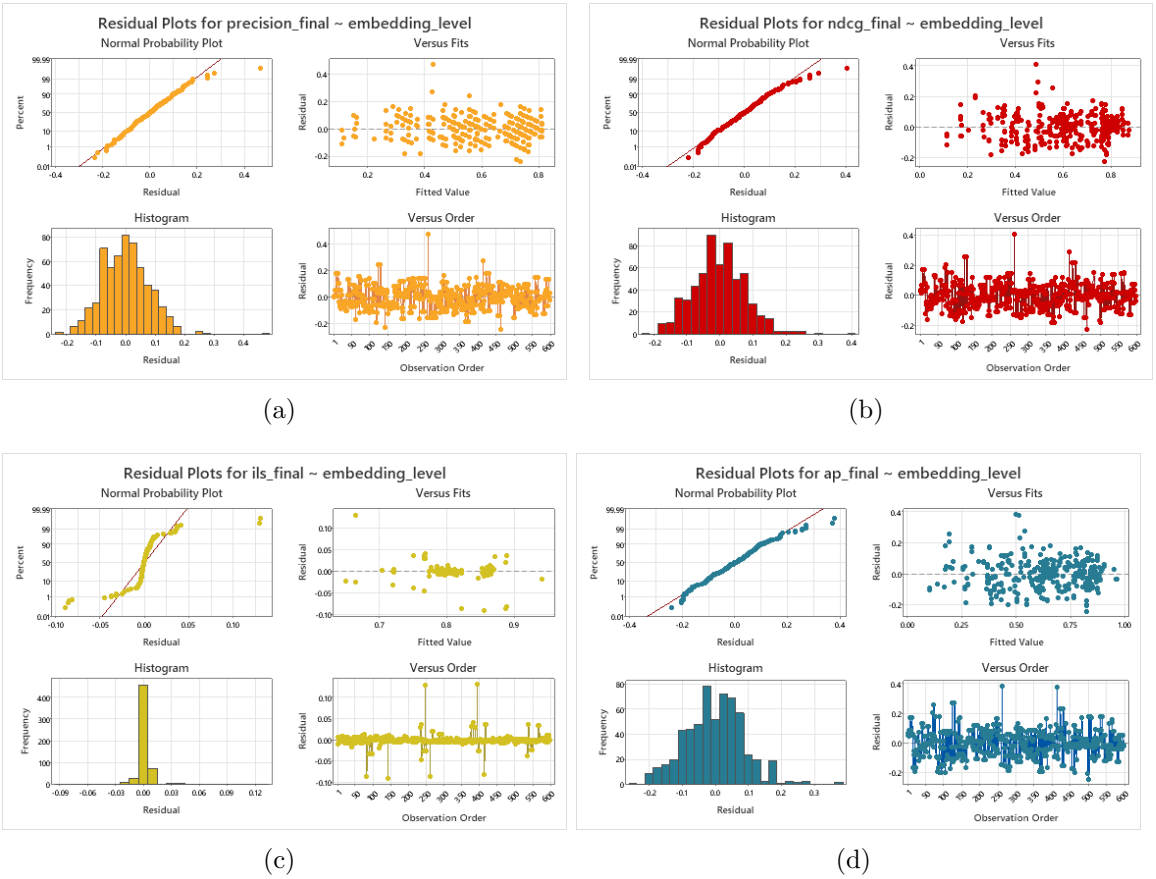


Figure 9. One-way ANOVA residual plots for metrics as an effect of embedding_level. All plots with the exception of (c) indicate that the sample is roughly NID.

Content Level	Precision	nDCG	ILS	MAP
1	0.54	0.556	0.857	0.575
2	0.584	0.614	0.803	0.641
3	0.581	0.619	0.804	0.654
4	0.583	0.615	0.787	0.649
p-value	< 0.01	< 0.01	$^*< 0.01$	< 0.01

Table 2. Mean metric values for different content levels. The model used is ChatGPT with *prompt_style*=‘zero‘ for $p = 1$ prompts. Scores are based on $k = 20$, $p = 1$ recommendations matched against the evaluation set. *P-value for ILS is determined via Kruskal-Wallis with tie-breaking. We find that content level does have a significant effect on each of the tested metrics at $\alpha = 0.01$

with ILS as DV is shown in Figure 10(b). Overall, our findings are 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 content level

Precision

embedding_level	N	Mean	Grouping
2	150	0.583667	A
4	150	0.582667	A
3	150	0.581333	A
1	150	0.539667	B

nDCG

embedding_level	N	Mean	Grouping
3	150	0.619086	A
4	150	0.614631	A
2	150	0.613715	A
1	150	0.555804	B

AP

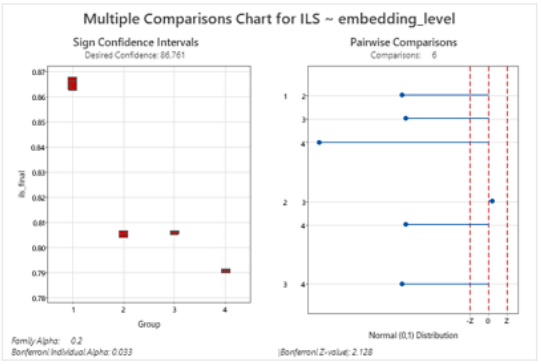
embedding_level	N	Mean	Grouping
3	150	0.653919	A
4	150	0.648556	A
2	150	0.640813	A
1	150	0.574945	B

Means that do not share a letter are significantly different.

(a)

ILS

Groups	Z vs. Critical value	P-value
1 vs. 4	19.5012 >= 2.128	0
3 vs. 4	9.9774 >= 2.128	0
1 vs. 2	9.9534 >= 2.128	0
2 vs. 4	9.5478 >= 2.128	0
1 vs. 3	9.5238 >= 2.128	0



(b)

Figure 10. TukeyHSD groupings (left) and Dunn's test (right) for embedding level effect on evaluation metrics. Tukey is based on 95% confidence. Results indicate that embedding content levels 2-4 are not significantly different, but interestingly increase metric scores over embedding level 1.

increases, performance also increases. This is in conflict with the preconceived notion 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 with in experimentation. We choose content level 4 embeddings due to a qualitative assessment that they showcase more reasonable similar items.

3 Analysis of Iterative Feedback

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. We vary the following parameters:

- $prompt\_style \in \{`zero`, `few`, `CoT`\}$
- $k \in \{5, 10\}$
- $p \in \{3, 5\}$

For comparison against the default direct recommendation task with $k = 20$ and $p = 1$ we also vary *prompt_style*. For few-shot and CoT prompting styles, we only utilize a single example of k recommendations. All together, there are a total of 15 total settings to compare. Since direct recommendation only ever uses the same k and p treatments and thus would be found to be colinear, we combine these two factors into a new factor $config$ to compare both archetypes of the system in the same test. We examine interactions through this engineered factor in the post-hoc tests. A two-way ANOVA is used to test for significance in the metrics and we again examine the residuals shown in Figure 11 to ensure that the test is valid. ILS is the only result that violates parametric assumptions, so we display the Kruskall-Wallis with tie-breaking p-values in Table 3 alongside the p-values for the ANOVA tests. We find that all factors are significant, as well as the interaction between *prompt_style* and $config$. The post-hoc tests performed in Figure 12, Figure 13, and Figure 14 indicate that

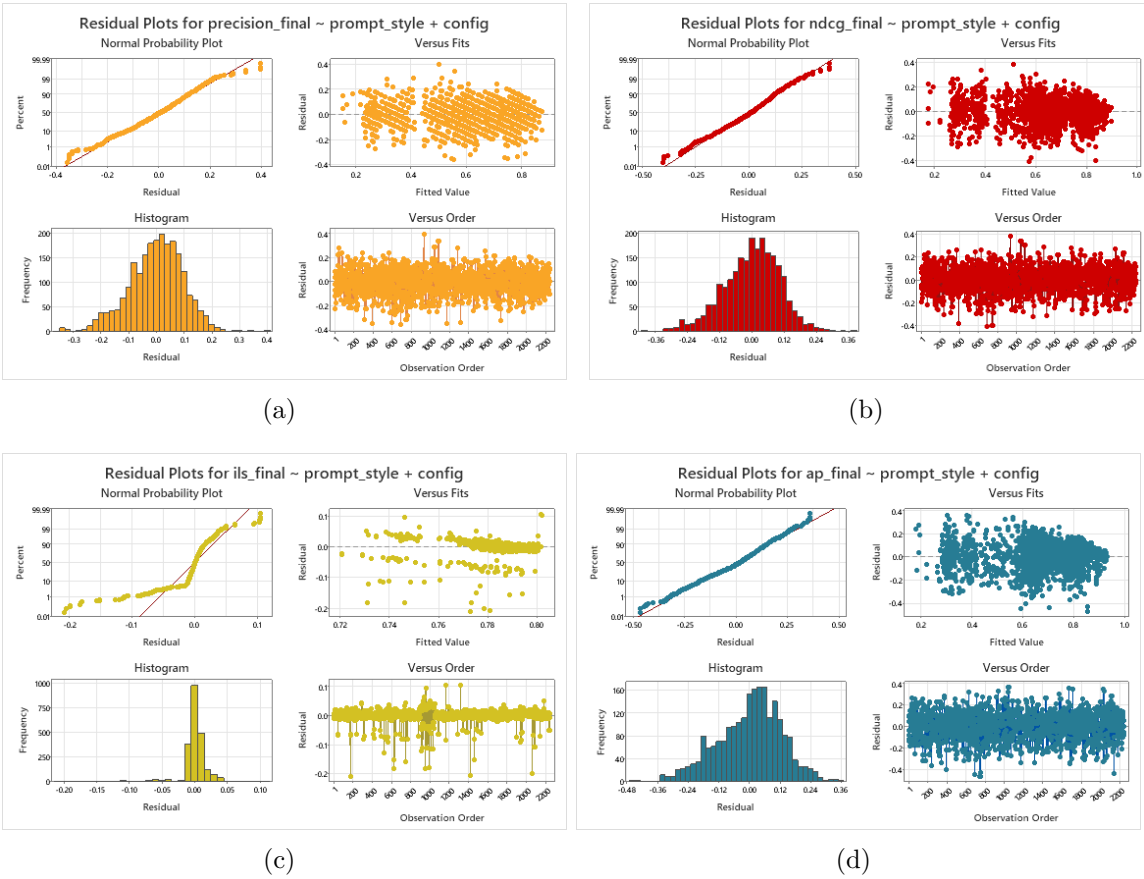
the configurations that allow for the most recommendations with $k = 10$ are shown to be significantly different from other levels for precision, nDCG, and AP. Few-shot and zero-shot are not significantly different, but both perform better than our CoT prompting across the board. Dunn's test results for ILS as DV in Figure 15 show 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 we choose as examples.

Interactions only show a notable difference between configurations for precision, but this provides evidence that reprompting is effective at 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 4 shows that the configuration $k = 10$, $p = 5$, and *prompt_style* = '*zero*' appears to be the best parameterization overall based on raw values. We further examine this model in Figure 16 to see how precision varies with coverage. 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.

Iterative Feedback Statistical Test p-values				
ANOVA				*Kruskal-Wallis*
Factor	Precision	nDCG	MAP	ILS
config	<0.01	<0.01	<0.01	<0.01
prompt_style	<0.01	<0.01	<0.01	<0.01
config*prompt_style	<0.01	<0.01	<0.01	-
uid	<0.01	<0.01	<0.01	-

Table 3. P-values for ANOVA and Kruskal-Wallis by factor and metric for evaluating iterative feedback. All factors are found to be significant at $\alpha = 0.01$



(a) (b)

(c) (d)

Figure 11. Two-way ANOVA residual plots for metrics as an effect of prompt style and configuration.

4 Analysis of ChatGPT as a Top-n Recommender

We compare 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-k recommenders)**. The two best parameterizations of the pipeline with and without reprompting, as indicated in Table 4, are selected to represent ChatGPT. We employ four total configurations that utilize NMF as the underlying recommender component.

NMF-item constructs a set of k_f recommendations for a user u by first building

Precision

config	N	Mean	Grouping		
k=10p=5	450	0.629789	A		
k=10p=3	450	0.618503	A	B	
k=5p=3	450	0.609146		B	
k=5p=5	450	0.608094		B	
k=20p=1	450	0.567736			C

config*prompt_style	N	Mean	Grouping			
k=10p=5 zero	150	0.636667	A			
k=10p=3 few	150	0.628877	A	B		
k=10p=5 few	150	0.627386	A	B		
k=10p=5 cot	150	0.625316	A	B	C	
k=10p=3 zero	150	0.619632	A	B	C	
k=5p=3 few	150	0.616000	A	B	C	
k=5p=3 cot	150	0.615175	A	B	C	
k=5p=5 few	150	0.611632	A	B	C	
k=5p=5 zero	150	0.610596	A	B	C	
k=10p=3 cot	150	0.607000	A	B	C	
k=5p=5 cot	150	0.602053	A	B	C	
k=20p=1 few	150	0.597000		B	C	
k=5p=3 zero	150	0.596263		B	C	
k=20p=1 zero	150	0.586193			C	
k=20p=1 cot	150	0.520015				D

prompt_style	N	Mean	Grouping		
few	750	0.616179	A		
zero	750	0.609870	A		
cot	750	0.593912		B	

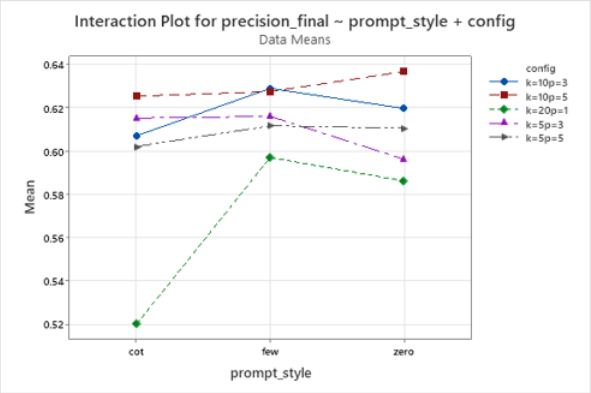


Figure 12. TukeyHSD groupings and interaction plot for precision as an effect of pipeline configuration and prompting style. Tukey is conducted at 95% confidence. Results show that reprompting does have a significant effect on the performance of the system.

individual lists of k_f unique recommendations for each positive item in the example set, creating a pool P_u of possible recommendations where $|P_u| = k_f * |\mathbb{E}_u|$. This pool is then reduced to the top k_f items with the highest similarity to the items in $\mathbb{E}_u$. **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,

nDCG

config	N	Mean	Grouping
k=10p=5	450	0.646070	A
k=10p=3	450	0.643220	A
k=5p=3	450	0.633576	A B
k=5p=5	450	0.621372	B
k=20p=1	450	0.590382	C

config*prompt_style	N	Mean	Grouping
k=10p=5 zero	150	0.656131	A
k=10p=3 zero	150	0.653291	A
k=5p=3 few	150	0.648346	A
k=10p=5 few	150	0.644929	A
k=10p=3 few	150	0.644758	A
k=10p=5 cot	150	0.637149	A
k=10p=3 cot	150	0.631610	A
k=5p=3 cot	150	0.627965	A
k=20p=1 few	150	0.626197	A
k=5p=3 zero	150	0.624417	A
k=5p=5 zero	150	0.623750	A
k=5p=5 few	150	0.620284	A
k=5p=5 cot	150	0.620082	A
k=20p=1 zero	150	0.617660	A
k=20p=1 cot	150	0.527289	B

prompt_style	N	Mean	Grouping
few	750	0.636903	A
zero	750	0.635050	A
cot	750	0.608819	B

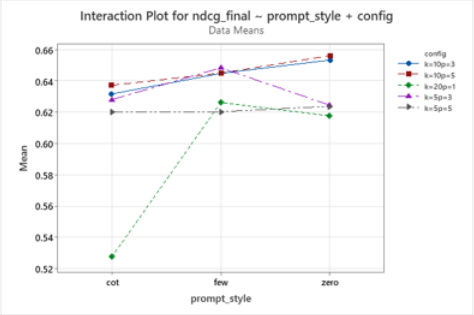


Figure 13. TukeyHSD groupings and interaction plot for nDCG as an effect of pipeline configuration and prompting style. Tukey is conducted at 95% confidence. Results indicate that model configurations using more prompts increases metric value, but interactions between config and *prompt_style* are largely insignificant.

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 correspondence is not likely to translate well 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 own learned embeddings for relevancy matching to give a more accurate assessment of both its performance and the overall effectiveness of the proposed evaluation approach.

The parameters for NMF are found by using grid search with a 5% validation split taken from the non-evaluation items. Training is carried out for 15,000 updates using

AP

config	N	Mean	Grouping		
k=10p=3	450	0.665800	A		
k=10p=5	450	0.662316	A		
k=5p=3	450	0.655402	A	B	
k=5p=5	450	0.636023		B	C
k=20p=1	450	0.614043			C

config*prompt_style	N	Mean	Grouping	
k=10p=3 zero	150	0.681927	A	
k=5p=3 few	150	0.675725	A	
k=10p=5 zero	150	0.673469	A	
k=10p=3 few	150	0.664086	A	
k=10p=5 few	150	0.660244	A	
k=20p=1 few	150	0.657598	A	
k=10p=5 cot	150	0.653236	A	
k=10p=3 cot	150	0.651386	A	
k=20p=1 zero	150	0.650141	A	
k=5p=3 zero	150	0.646764	A	
k=5p=3 cot	150	0.643716	A	
k=5p=5 cot	150	0.638240	A	
k=5p=5 zero	150	0.638056	A	
k=5p=5 few	150	0.631772	A	
k=20p=1 cot	150	0.534390		B

prompt_style	N	Mean	Grouping	
zero	750	0.658071	A	
few	750	0.657885	A	
cot	750	0.624194		B

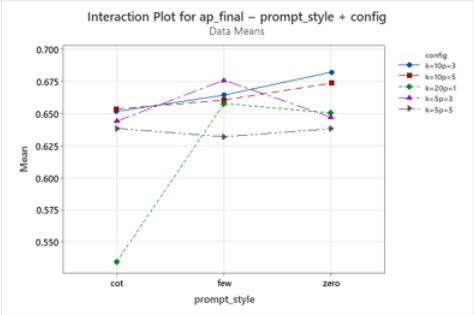


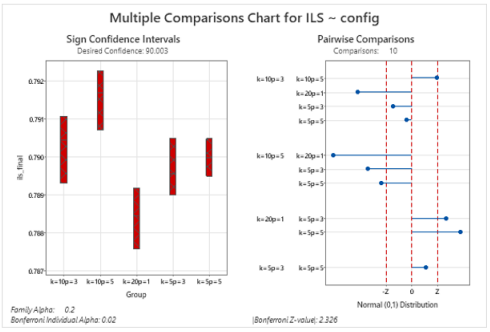
Figure 14. TukeyHSD groupings and interaction plot for AP as an effect of pipeline configuration and prompting style. Tukey is conducted at 95% confidence. This shows similar results to nDCG for the same factors. Utilizing more prompts is again found to be a significant effect, but the interaction is not.

SGD, with model parameter restoration based on maximum RMSE for the validation set. We find the optimal parameter set as $\lambda = 0.05$, $\alpha = 1.2$, and $d = 50$ using this approach. Results of evaluating the NMF model on the interactions in the evaluation split $\mathbb{T}$ can be seen in Table 5.

The mean metric values for each model tested can be seen in Table 6, along with the p-values for the statistical tests. Post-hoc tests were performed but have been omitted for space. 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

ILS

Groups	Z vs. Critical value	P-value
k=10p=5 vs. k=20p=1	7.05699 >= 2.326	0.0000
k=10p=3 vs. k=20p=1	4.80131 >= 2.326	0.0000
k=20p=1 vs. k=5p=5	4.36275 >= 2.326	0.0000
k=10p=5 vs. k=5p=3	3.95238 >= 2.326	0.0001
k=20p=1 vs. k=5p=3	3.10461 >= 2.326	0.0019
k=10p=5 vs. k=5p=5	2.69424 >= 2.326	0.0071



Groups	Z vs. Critical value	P-value
cot vs. few	10.9642 >= 1.834	0
cot vs. zero	10.4012 >= 1.834	0

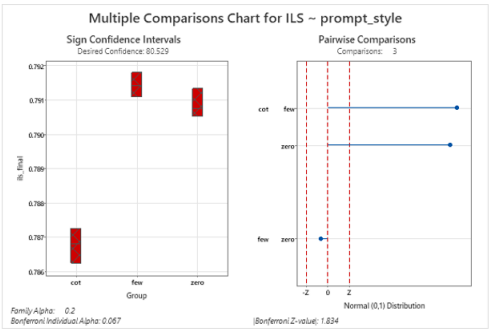
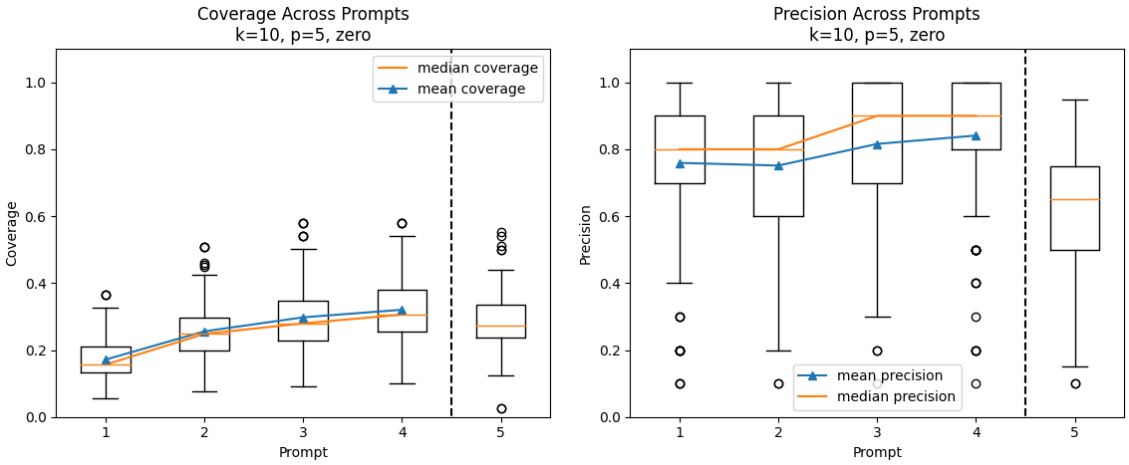


Figure 15. Dunn's test results on ILS as an effect of prompt style and pipeline config. Results show that generally increasing the number of prompts raises the similarity of items in the recommendation list.



(a) (b)

Figure 16. Coverage and precision distributions for different prompt numbers using best configuration from Table 4. We see here that ChatGPT contains to match unique items in the feedback set while further increasing precision.

Prompt Style	k	p	Precision	nDCG	ILS	MAP	UR
Zero-Shot	20	1	.586	.618	.791	0.65	6e-4
	5	3	.593	.624	0.79	.647	5e-4
		5	.611	.624	.788	.638	1e-3
	10	3	.612	.653	.789	.682	1e-3
		5	**.637**	**.656**	.791	.674	1e-3
Few-Shot	20	1	.597	.626	.792	.658	6e-4
	5	3	.616	.648	.788	**.676**	9e-4
		5	.612	.62	.783	.632	2e-3
	10	3	.629	.645	.787	.664	1e-3
		5	.627	.645	.788	.66	5e-3
CoT	20	1	.52	.527	**.743**	.534	1e-2
	5	3	.615	.628	.784	.644	3e-3
		5	.602	.62	.787	.638	2e-3
	10	3	.607	.632	.786	.651	6e-3
		5	.625	.637	.783	.653	1e-2

Table 4. 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. Unmatched Ratio (UR) indicates the average fraction of all recommended items that were lost due to matching issues. Best results are colored in each column.

RMSE	Precision@5	Recall@5	MAP@5	nDCG@5
1.181	0.796	0.052	0.672	0.553

Table 5. Average NMF performance as measured on the evaluation set outside of the proposed methodology.

evaluated using their own learned embeddings perform similarly to ChatGPT. This may indicate that ChatGPT with iterative feedback is as effective as a supervised model in the eyes of our evaluation pipeline. However, we refrain from giving too much significance to this claim without testing more model varieties. Based on these findings, we answer **RQ2** by showcasing that ChatGPT is at the very least preferable to random selection at recommendation.

Model	Precision	nDCG	ILS	MAP	UR
ChatGPT (k=10, p=5, zero)	.637	.656	.791	.674	1e-3
ChatGPT (k=20, p=1, few)	.597	.626	.792	.658	6e-4
NMF-item	.263	.262	.773	.259	-
NMF-user	.27	.285	.77	.3	-
Random	.243	.246	.76	.246	-
p-values	< 0.01	< 0.01	$^*< 0.01$	< 0.01	
NMF-item (learned embeddings)	.626	.629	.639	.634	-
NMF-user (learned embeddings)	.646	.648	.637	.647	-

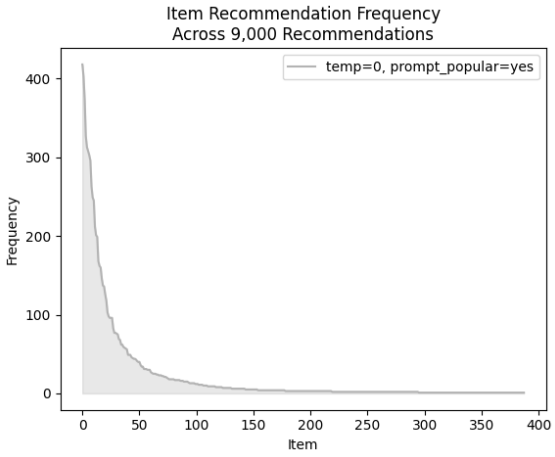
Table 6. Comparison of two best pipeline parameterizations (with and without reprompting) against baseline models Non-negative Matrix Factorization (NMF) and Random. Metric values are averaged. *P-value for ILS is determined via Kruskal-Wallis with tie-breaking.

5 Exploring Popularity Bias in Recommendations

Our last experiment tries to answer **RQ3: Does Chat-GPT exhibit popularity bias in recommendation?** Due to the amount of tests performed, it is apparent that ChatGPT prefers certain recommendations over others; indicating popularity bias. The configuration $k = 10$, $p = 5$ produces the largest quantity of recommendations of the settings tested, so we visualize how frequently items are recommended across 50 users for 3 replicates in Figure 17(a). Figure 17(b) shows the most common recommendations, which appear to strongly coincide with entries on the IMDb top 250 movies [46].

With this evidence alone **RQ3** could easily be answered. However, we would also like to explore whether there is a way that this could be mitigated. We choose the following factors and levels to test this:

- $prompt\_popular \in \{`no`, `yes`\}$
- $temperature \in \{0, 0.5, 1\}$



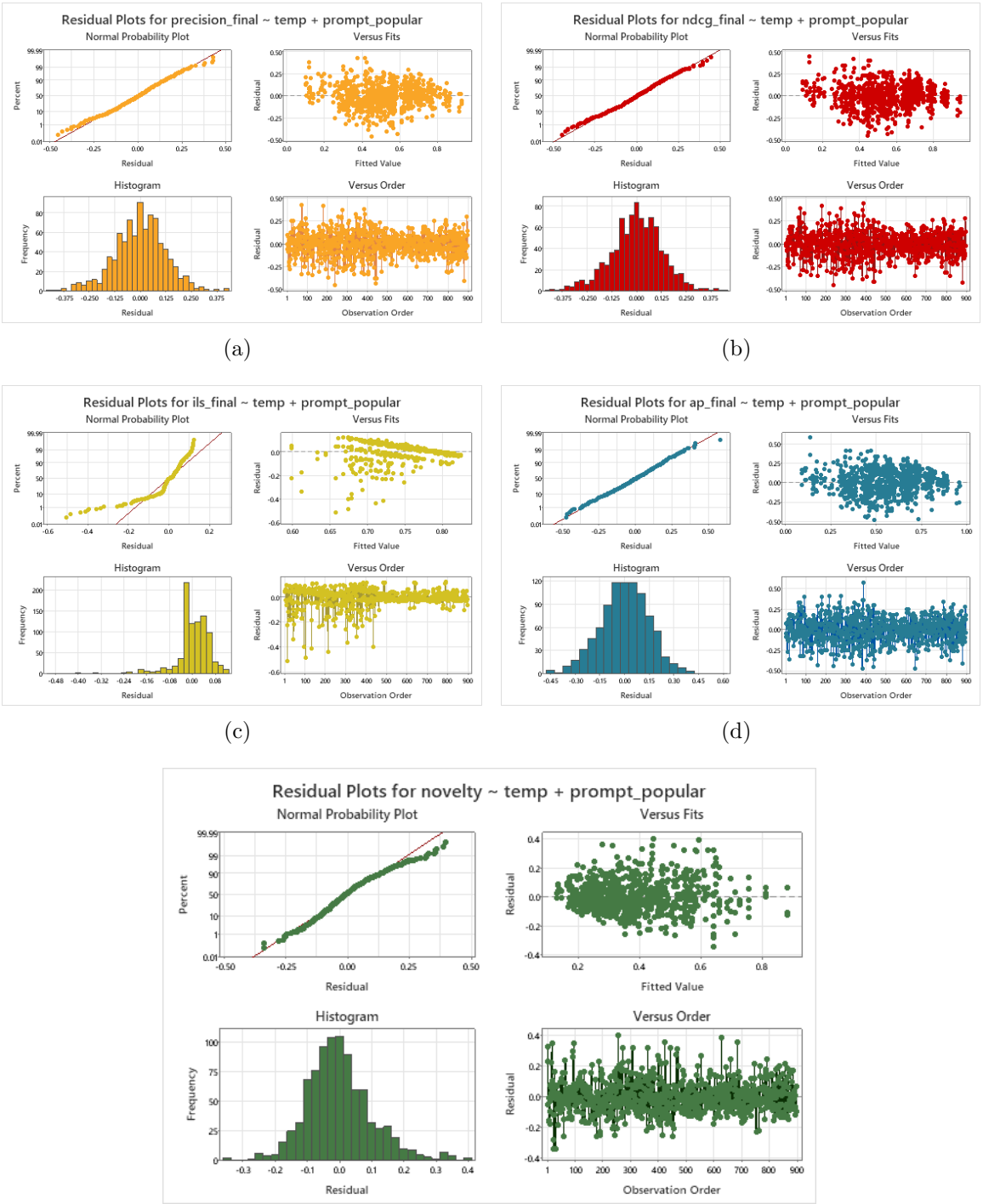
(a)

1. 'The Shawshank Redemption (1994)': 418
2. 'The Departed (2006)': 403
3. 'The Prestige (2006)': 374
4. 'Fight Club (1999)': 327
5. 'The Sixth Sense (1999)': 313
6. 'The Silence of the Lambs (1991)': 308
7. 'The Green Mile (1999)': 303
8. 'The Truman Show (1998)': 296
9. 'The Matrix (1999)': 263
10. 'The Dark Knight (2008)': 249
11. 'Inception (2010)': 245
12. 'The Usual Suspects (1995)': 212
13. 'Pulp Fiction (1994)': 201
14. 'Memento (2000)': 199
15. 'The Godfather (1972)': 168

(b)

Figure 17. Frequency of recommendations across 9,000 total recommendation instances.

The factor *prompt_popular* indicates whether we allow ChatGPT to use popular recommendations. When *prompt_popular*=‘no‘ we merely add the additional instruction: ”Try to recommend movies that are less popular,” to the initial prompt and reprompts. The default settings are *prompt_popular* = ‘*yes*‘ and *temperature* = 0. To determine whether these factors play a role in reducing popularity bias in recommendations, we are mostly concerned with the novelty of the recommendations. However, we performed significance tests for all other metrics as before. The residuals of the ANOVAs are shown in Figure 18, which shows similar findings to previous experiments. The P-values for the statistical tests are shown in Tab 7. We find that all individual effects and interactions are significant for novelty, but only *prompt_popular* is significant for other metrics. TukeyHSD comparison on *prompt_popular* in Figure 19(a) shows a significant performance reduction by restricting the recommendation of popular items. Figure 19(b) shows that as *temperature* increases, the ILS of the



(a) (b)

(c) (d)

(e)

Figure 18. Two-way ANOVA residual plots for metrics as an effect of *temp* and *prompt_popular*. All plots with the exception of (c) indicate that the sample is roughly NID.

recommended items decreases considerably.

Popularity Statistical Test p-values					
ANOVA					*Kruskal-Wallis*
Factor	Precision	nDCG	MAP	Novelty	ILS
temperature	.174	.246	.242	<.01	<.01
prompt_popular	<.01	<.01	<.01	<.01	<.01
temperature * prompt_popular	.908	.931	.927	<.01	-
uid	<.01	<.01	<.01	<.01	-

Table 7. P-values for ANOVA and Kruskal-Wallis by factor and metric for evaluating recommendation popularity. We find that only *prompt_popular* is significant for standard metrics, but both main effects and the interaction is significant for novelty at $\alpha = 0.01$

The main results of interest are how these factors influence novelty. TukeyHSD comparison for novelty in Figure 20 indicates 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 *prompt_popular*='no'. The effect of this is better seen in Figure 21, clearly indicating a reduction of the short-tail in item frequency. One caveat to this is that using these options comes at the cost of performance; as indicated by analysis of the other metrics. However, this is a common trade-off associated with traditional approaches to improve variety in recommendation [47][48].

Lastly, we summarize the means of the results in Table 8 for ease of comparison. We further note that increasing *temperature* and restricting popular recommendations has the undesirable effect of increasing the quantity of unmatchable items.

6 Chapter Summary

These experiments have provided answers to each of our three research questions:

Precision

prompt_popular	N	Mean	Grouping
yes	450	0.634132	A
no	450	0.426342	B

nDCG

prompt_popular	N	Mean	Grouping
yes	450	0.651516	A
no	450	0.434659	B

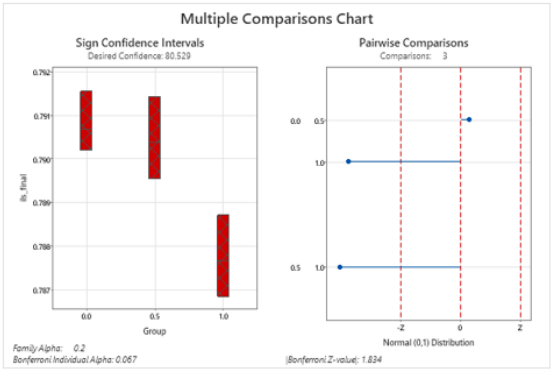
AP

prompt_popular	N	Mean	Grouping
yes	450	0.665468	A
no	450	0.444552	B

(a)

ILS

temperature Groups	Z vs. Critical value	P-value
0.5 vs. 1.0	3.67994 >= 1.834	0.0002
0.0 vs. 1.0	3.41736 >= 1.834	0.0006



(b)

Figure 19. TukeyHSD (a) for ILS as an effect of *prompt_popular* and Dunn's test for ILS as an effect of *temp*. Tukey test conducted with 95% confidence.

Novelty

temperature	N	Mean	Grouping
1.0	300	0.468970	A
0.5	300	0.347688	B
0.0	300	0.310108	C

prompt_popular	N	Mean	Grouping
no	450	0.441879	A
yes	450	0.309299	B

temperature*prompt_popular	N	Mean	Grouping
1.0 no	150	0.580527	A
0.5 no	150	0.407220	B
1.0 yes	150	0.357413	C
0.0 no	150	0.337889	C
0.5 yes	150	0.288157	D
0.0 yes	150	0.282328	D

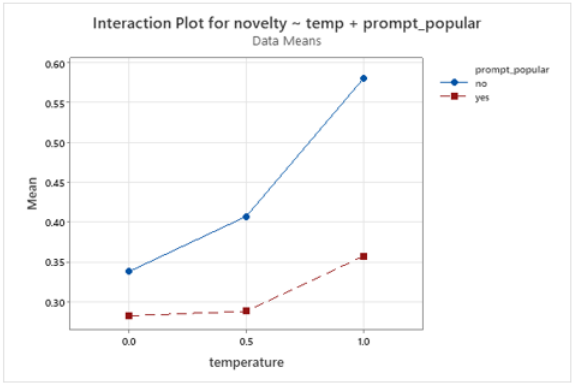
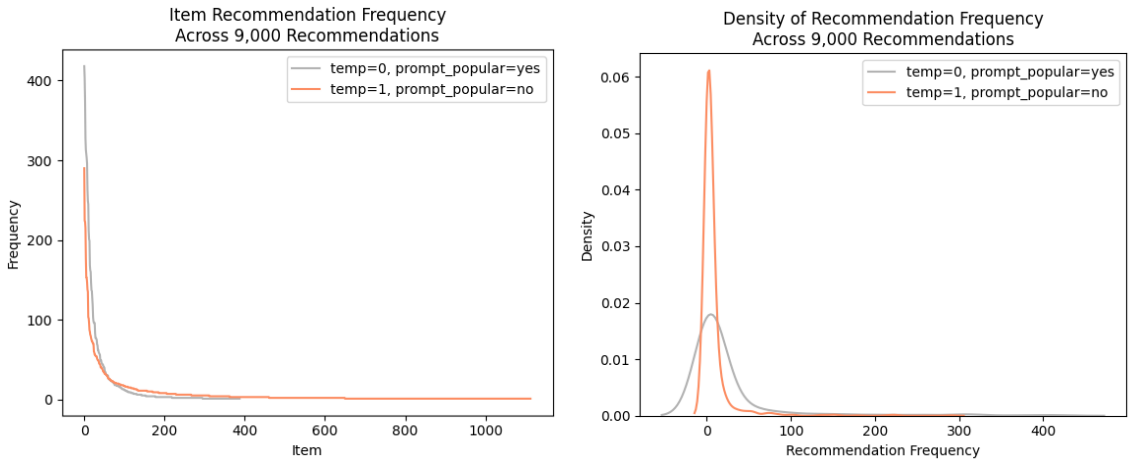


Figure 20. TukeyHSD groupings and interaction plot for Novelty as an effect of *temp* and *prompt_popular*. Tukey is conducted at 95% confidence.

- **RQ1**: How does the ability to converse impact recommendation in language models?



(a) (b)

Figure 21. Recommendation frequency by item and and density of two ChatGPT configurations. The model used has parameters $p = 5$, $k = 10$, *prompt_style* =‘zero.‘ Prompting with *temperature* = 1 and specifying that popular recommendations should be limited (*prompt_popular*=no) reduces the short-tail of recommendation frequency.

Temp	Prompt Popular	Precision	nDCG	ILS	MAP	Novelty	UR
0.0	no	.43	.438	.725	.448	.338	3e-2
	yes	.637	.656	.791	.674	.282	1e-3
0.5	no	.437	.444	.747	.454	.407	3e-2
	yes	.64	.656	.792	.67	.288	2e-3
1.0	no	.413	.422	.712	.43	.581	6e-2
	yes	.626	.643	.788	.65	.357	5e-3

Table 8. Mean metric values for combinations of *temp* and *prompt_popular* parameters.

- **RQ2**: How do language models perform at recommendation in their *typical* use-case? (as primarily item-based, top-k recommenders)

- **RQ3**: Does Chat-GPT exhibit popularity bias in recommendation?

We answer **RQ1** by showing that model configurations with multiple prompts score higher in precision than models using only a single prompt. Indicating that we are able to inject additional information mid-conversation to help guide ChatGPT

towards a better answer.

We answer **RQ2** by showing that ChatGPT is a significantly better recommender than a random baseline. While we find interesting results that imply ChatGPT with reprompting performs nearly identically to a supervised model based on our evaluation pipeline, we cannot make claims that it is actually comparable in performance to any supervised method.

We answer **RQ3** by showing that ChatGPT clearly exhibits popularity bias in its recommendations. We propose to use *temperature* and a prompt-based restriction of popular recommendations to increase the novelty of recommendations.

CHAPTER V

CONCLUSIONS

In this thesis we constructed an evaluation pipeline around ChatGPT to evaluate it as a conversational, direct top-n recommendation system. Compared to other studies that have tested ChatGPT's ability to pull the best recommendation out of a pool of candidates [5][6], we sought to evaluate ChatGPT in way that is more consistent with how a user would interact with it for recommendation. In seeking this goal, we were able to answer all three of our research questions. We were able to show that reprompting ChatGPT with feedback mid-conversation does have a significant effect on the performance of the system. Additionally, we showed that ChatGPT is significantly better than a random recommender, indicating that its domain knowledge is useful for recommendation. Lastly, we were able to evaluate popularity bias in ChatGPT, and find possible ways to mitigate it to receive more novel recommendations.

Despite these satisfying findings, our system does have several limitations. First, we are dependent on a relatively large amount of text data in order to produce robust content embeddings. Additionally, we are constrained to relatively old movie releases from our dataset. ChatGPT has been trained on data up to September 2021, so we are omitting 10 years worth of extra items from our evaluation. Future work may

require that a more recent dataset be used in order to fully capture how ChatGPT performs on recent information.

Another limitation comes from our lack of comparable models to compare to ChatGPT in the pipeline. In a follow up study, it would be more appropriate to compare ChatGPT's performance against other LLM, as many other authors have done [4], [6].

Recommendation is a task that only becomes more important with each passing year. As the amount of content available online continues to increase, it may soon prove to be a more efficient use of time to have language models peruse this content in your place. Overall, we believe that user experience is part of the recommendation process as well. If the user is now able to have a consistent part in what they choose to see and consume, this might help to eliminate many of the problem that plague recommendation systems today; like popularity bias.

REFERENCES

[1] C. A. Thompson, M. H. Goker, and P. Langley, "A personalized system for conversational recommendations", *Journal of Artificial Intelligence Research*, vol. 21, pp. 393–428, 2004.

[2] M. Qiu, F.-L. Li, S. Wang, *et al.*, "Alime chat: A sequence to sequence and rerank based chatbot engine", in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2017, pp. 498–503.

[3] L. Ouyang, J. Wu, X. Jiang, *et al.*, *Training language models to follow instructions with human feedback*, 2022. arXiv: 2203.02155 [cs.CL].

[4] Y. Gao, T. Sheng, Y. Xiang, Y. Xiong, H. Wang, and J. Zhang, "Chat-rec: Towards interactive and explainable llms-augmented recommender system", *arXiv preprint arXiv:2303.14524*, 2023.

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

[6] W.-C. Kang, J. Ni, N. Mehta, *et al.*, "Do llms understand user preferences? evaluating llms on user rating prediction", *arXiv preprint arXiv:2305.06474*, 2023.

[7] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification", in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 1422–1432.

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

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] K. Cho, B. van Merriënboer, C. Gulcehre, *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. [Online]. Available: https://aclanthology.org/D14-1179.

[11] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks", *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997. DOI: 10.1109/78.650093.

[12] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training", 2018.

[13] B. Liu *et al.*, *Web data mining: exploring hyperlinks, contents, and usage data*. Springer, 2011, vol. 1.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space", *arXiv preprint arXiv:1301.3781*, 2013.

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality", *Advances in neural information processing systems*, vol. 26, 2013.

[16] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation", in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[17] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473 [cs.CL].

[18] M.-T. Luong, H. Pham, and C. D. Manning, *Effective approaches to attention-based neural machine translation*, 2015. arXiv: 1508.04025 [cs.CL].

[19] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need", *Advances in neural information processing systems*, vol. 30, 2017.

[20] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners", *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[21] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners", *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[22] J. Wei, X. Wang, D. Schuurmans, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models", *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[23] P. Lops, M. de Gemmis, and G. Semeraro, "Content-based recommender systems: State of the art and trends", in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Boston, MA: Springer US, 2011, pp. 73–105, ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_3. [Online]. Available: https://doi.org/10.1007/978-0-387-85820-3_3.

[24] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms", in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 285–295.

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

[26] R. Mehta and K. Rana, "A review on matrix factorization techniques in recommender systems", in *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*, IEEE, 2017, pp. 269–274.

[27] Y. Zhang, H. Ding, Z. Shui, *et al.*, "Language models as recommender systems: Evaluations and limitations", 2021.

[28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", *arXiv preprint arXiv:1810.04805*, 2018.

[29] I. Cantador, P. Brusilovsky, and T. Kuflik, "Second workshop on information heterogeneity and fusion in recommender systems (hetrec2011)", in *Proceedings of the fifth ACM conference on Recommender systems*, 2011, pp. 387–388.

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

[31] *Imdb non-commercial datasets: Title.basics*, `https://datasets.imdbws.com/title.basics.tsv.gz`, Accessed: 2023-08-01.

[32] *Rotten tomatoes*, `https://www.rottentomatoes.com/`, Accessed: 2023-08-01.

[33] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering", in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 1999, pp. 230–237.

[34] D. Cer, Y. Yang, S.-y. Kong, *et al.*, "Universal sentence encoder", *arXiv preprint arXiv:1803.11175*, 2018.

[35] Z. Lin, M. Feng, C. N. d. Santos, *et al.*, "A structured self-attentive sentence embedding", *arXiv preprint arXiv:1703.03130*, 2017.

[36] X. Ye and G. Durrett, "The unreliability of explanations in few-shot in-context learning", *arXiv preprint arXiv:2205.03401*, 2022.

[37] Y. Fu, H. Peng, A. Sabharwal, P. Clark, and T. Khot, "Complexity-based prompting for multi-step reasoning", *arXiv preprint arXiv:2210.00720*, 2022.

[38] L. Yujian and L. Bo, "A normalized levenshtein distance metric", *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[39] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems", *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 5–53, 2004.

[40] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques", *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422–446, 2002.

[41] M. Jesse, C. Bauer, and D. Jannach, "Intra-list similarity and human diversity perceptions of recommendations: The details matter", *User Modeling and User-Adapted Interaction*, pp. 1–34, 2022.

[42] M. Kaminskas and D. Bridge, "Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems", *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 7, no. 1, pp. 1–42, 2016.

[43] R. Horsley, *Randomized complete block design (rcbd)*, `https://www.ndsu.edu/faculty/horsley/RCBD_revised_notes.pdf`, Accessed: 2023-08-07.

[44] *Post hoc definition and types of tests*, `https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/post-hoc/`, Accessed: 2023-08-01.

[45] B. G. Tabachnick and L. S. Fidell, *Experimental designs using ANOVA*. Thomson/Brooks/Cole Belmont, CA, 2007, vol. 724.

[46] *Imdb top 250 movies*, `https://www.imdb.com/chart/top/`, Accessed: 2023-08-01.

[47] L. Iaquinta, M. De Gemmis, P. Lops, G. Semeraro, M. Filannino, and P. Molino, "Introducing serendipity in a content-based recommender system", in *2008 eighth international conference on hybrid intelligent systems*, IEEE, 2008, pp. 168–173.

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

CURRICULUM VITAE

NAME: Kyle Dylan Spurlock

ADDRESS: Computer Engineering & Computer Science Department
Speed School of Engineering
University of Louisville
Louisville, KY 40292

EDUCATION:

M.S., Computer Science & Engineering
January 2022 - August 2023
GPA: 4.0
University of Louisville, *Louisville, Kentucky*

B.S., Computer Science
August 2018 - December 2021
Morehead State University, *Morehead, Kentucky, USA*

EMPLOYMENT:

Graduate Teaching Assistant
May 2022 - Present
University of Louisville, *Louisville, Kentucky, USA*

Grader
January 2022 - May 2022
University of Louisville, *Louisville, Kentucky, USA*

NSF REU Researcher
May 2021 - August 2021
University of Louisville, *Louisville, Kentucky, USA*

Undergraduate Researcher
January 2020 - December 2021
Morehead State University, *Morehead, Kentucky, USA*

PUBLICATIONS:

1. Spurlock, K., & Elgazzar, H. (2020, October). Predicting COVID-19 infection groups using social networks and machine learning algorithms. In 2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON) (pp. 0245-0251). IEEE.

2. Elgazzar, H., Spurlock, K., & Bogart, T. (2021). Evolutionary clustering and community detection algorithms for social media health surveillance. Machine Learning with Applications, 6, 100084.

3. Spurlock, K., & Elgazzar, H. (2022). A genetic mixed-integer optimization of neural network hyper-parameters. The Journal of Supercomputing, 78(12), 14680-14702.

ACHIEVEMENTS AND AWARDS:

- J.B. Speed School of Engineering - Alfred T. Chen Scholarship (2023)
- J.B. Speed School of Engineering - Department of Computer Engineering and Computer Science Alumni Scholarship (2023)
- Morehead State University - Outstanding Computer Science Student (2022)