arXiv:2411.19352v1 [cs.AI] 28 Nov 2024

OMuleT: Orchestrating Multiple Tools for Practicable Conversational Recommendation

Se-eun Yoon
University of California, San Diego
La Jolla, CA, USA
seeuny@ucsd.edu

Xiaokai Wei
Roblox
San Mateo, CA, USA
xwei@roblox.com

Yexi Jiang
Roblox
San Mateo, CA, USA
yjiang@roblox.com

Rachit Pareek
Roblox
San Mateo, CA, USA
rpareek@roblox.com

Frank Ong
Roblox
San Mateo, CA, USA
fong@roblox.com

Kevin Gao
Roblox
San Mateo, CA, USA
kgao@roblox.com

Julian McAuley
University of California, San Diego
La Jolla, CA, USA
jmcauley@ucsd.edu

Michelle Gong
Roblox
San Mateo, CA, USA
mgong@roblox.com

Abstract

In this paper, we present a systematic effort to design, evaluate, and implement a realistic conversational recommender system (CRS). The objective of our system is to allow users to input free-form text to request recommendations, and then receive a list of relevant and diverse items. While previous work on synthetic queries augments large language models (LLMs) with 1-3 tools, we argue that a more extensive toolbox is necessary to effectively handle real user requests. As such, we propose a novel approach that equips LLMs with over 10 tools, providing them access to the internal knowledge base and API calls used in production. We evaluate our model on a dataset of real users and show that it generates relevant, novel, and diverse recommendations compared to vanilla LLMs. Furthermore, we conduct ablation studies to demonstrate the effectiveness of using the full range of tools in our toolbox. We share our designs and lessons learned from deploying the system for internal alpha release. Our contribution is the addressing of all four key aspects of a practicable CRS: (1) real user requests, (2) augmenting LLMs with a wide variety of tools, (3) extensive evaluation, and (4) deployment insights.

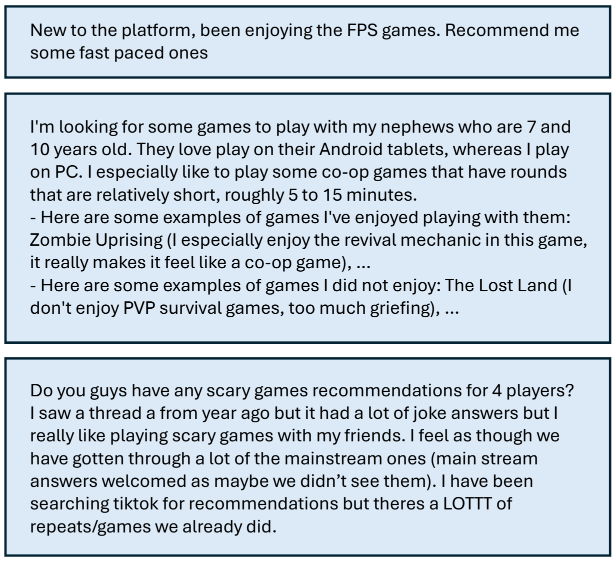
CCS Concepts

• **Information systems** → **Information retrieval**; **Users and interactive retrieval**; **Web applications**.

Keywords

conversational recommender systems, large language models

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.
Conf, Month DD, YYYY, City, State
© 2024 ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/tbd



New to the platform, been enjoying the FPS games. Recommend me some fast paced ones

I'm looking for some games to play with my nephews who are 7 and 10 years old. They love play on their Android tablets, whereas I play on PC. I especially like to play some co-op games that have rounds that are relatively short, roughly 5 to 15 minutes.
- Here are some examples of games I've enjoyed playing with them: Zombie Uprising (I especially enjoy the revival mechanic in this game, it really makes it feel like a co-op game), ...
- Here are some examples of games I did not enjoy: The Lost Land (I don't enjoy PVP survival games, too much griefing), ...

Do you guys have any scary games recommendations for 4 players? I saw a thread a from year ago but it had a lot of joke answers but I really like playing scary games with my friends. I feel as though we have gotten through a lot of the mainstream ones (main stream answers welcomed as maybe we didn't see them). I have been searching tiktok for recommendations but theres a LOTTT of repeats/games we already did.

Figure 1: Examples of recommendation requests from users.

ACM Reference Format:
Se-eun Yoon, Xiaokai Wei, Yexi Jiang, Rachit Pareek, Frank Ong, Kevin Gao, Julian McAuley, and Michelle Gong. 2024. OMuleT: Orchestrating Multiple Tools for Practicable Conversational Recommendation. In *Proceedings of Conf*. ACM, New York, NY, USA, 10 pages. https://doi.org/tbd

1 Introduction

Imagine a user who wants to find new games but faces thousands to millions of options. Since trying out various games can be time-consuming, one may want to get recommendations simply by saying in natural language what they want to play. Examples of such user requests are depicted in Figure 1, where users express their unique needs through diverse expressions. A conversational recommender system (CRS) that can take in such free-form requests and retrieve the most relevant items would greatly improve user experience in navigating through a vast choice of content.

While there are many works on CRS [3, 18, 22, 38, 43, 55, 56], rarely do we see a system in practice. Even though large language models (LLMs) have been demonstrated to be effective in conversational movie recommendation [11, 33], LLMs alone cannot be directly applied to many industrial domains. One limitation of LLMs is their dependence on fixed parameters, which restricts their ability to handle a dynamic pool of items and integrate up-to-date world knowledge without the costly process of fine-tuning. Furthermore, LLMs exhibit high popularity bias, frequently recommending or addressing the most well-known items [11].

This work contributes to practicable CRS research through the following efforts. First, we collect a dataset of *real user requests* and recommendations. This distinguishes our work from papers that use synthetic queries generated from traditional user-item interactions [13, 16, 44]. Real user requests are more challenging to process than requests synthesized from templates due to their variety, unstructured nature, and subjective language [11, 51]. Second, in order to process such complex requests, we argue that *a much larger number of tools* are required to augment LLMs for recommendations, compared to existing approaches that address synthetic queries with only 1-3 tools. For example, in real user requests, free-form casual utterances (e.g., using 'ptfs' to refer to the game 'Pilot Training Flight Simulator') require specialized tools for processing, which is not necessary for synthetic requests that use clearly defined item names. Another example is handling complex conditions, such as a user who plays games on a PC and wants games to play with 7- and 10-year-old nephews who use tablets, and providing a list of liked and disliked games and reasons (see Figure 1). Using just a search API [4] or a lookup API [19] may be insufficient for handling such conditions; multiple tools are required to address factors such as games popular among age groups, device compatibility, and similar games search. While a large number of tools may initially seem daunting to implement, our tools are relatively generic (e.g., unlike tools that require reviews [16]) and can be easily constructed from databases and APIs available in many industry settings.

Third, we propose OMuleT (Orchestrating Multiple Tools), a framework for augmenting LLMs with diverse tools to meet complex requests. Our method translates a user's raw utterance into a formatted intent, applies a tool-execution policy, and then augments the results to the LLM generating the recommendations (see Figure 3). This approach not only makes the system transparent and controllable, but it is more effective in performance than methods where an LLM generate its own tool execution policy [13, 41]. Finally, we perform *extensive evaluation* on two LLMs (LLaMA-405B [1] and GPT-4o [29]) and 8 metrics covering factuality, relevance, novelty, and diversity. Our results show that using our framework is more effective than baseline LLMs, and multiple tools are necessary for the best performance. We implement our model for internal testing and share our insights for deployment. To the best of our knowledge, we are the first work to address all the following elements that are essential for a practicable CRS:

- **Real user requests.** We use real user requests, which are more complex and diverse than queries synthesized from templates.
- **Framework for augmenting LLMs with nontrivial amount of tools.** Complex requests require the use of a wide variety of tools. Our tools are simple and generic, and our framework effectively orchestrates the tools to augment LLMs.
- **Extensive evaluation.** We conduct evaluations and ablation studies on various metrics and baselines to demonstrate the effectiveness of our approach.
- **Deployment insights.** We share our designs and lessons learned from deploying the system for internal demonstrations, hoping to provide guidance for practitioners.

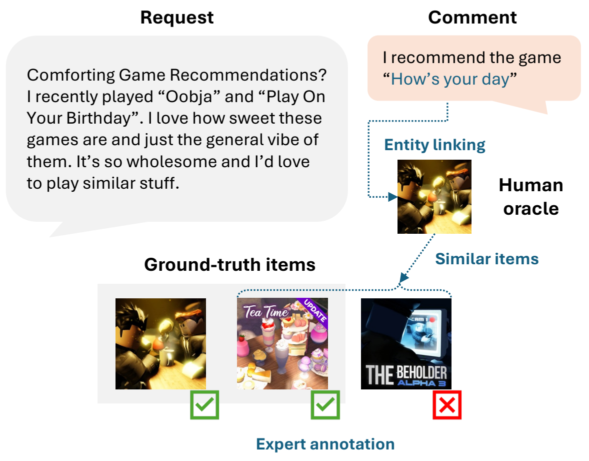

Request
Comforting Game Recommendations? I recently played "Oobja" and "Play On Your Birthday". I love how sweet these games are and just the general vibe of them. It's so wholesome and I'd love to play similar stuff.
Comment
I recommend the game "How's your day"
Entity linking
Human oracle
Similar items
Ground-truth items
Expert annotation

Figure 2: Our dataset collection process.

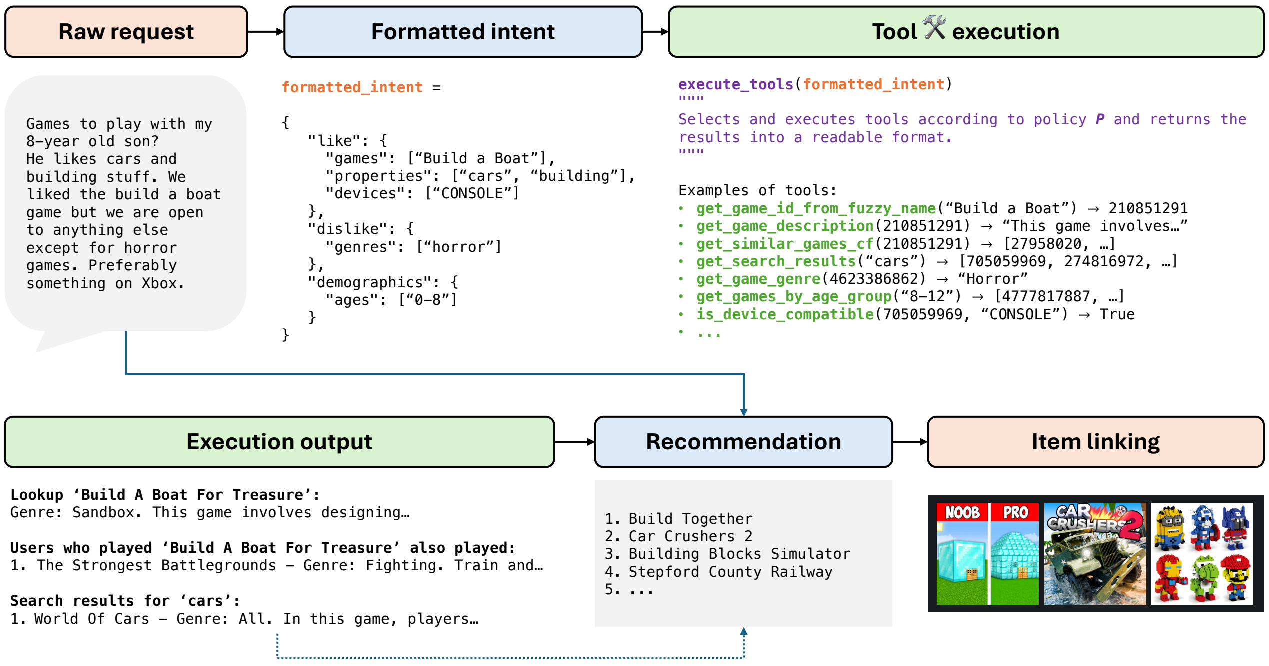
2 Problem Formulation

Given a user's recommendation request in free-form natural language, the agent should return a list of k items. The success of the task is measured by multiple criteria. First, the items should be **relevant** to the request; they should be what the user is asking for. An ideal approach to evaluate relevance is to get direct feedback from the user who made the request. However, in the early stages of model development, obtaining feedback for each iteration is impractical. Thus, we construct an evaluation data as a proxy for relevance, which we discuss in Section 3.1. Another important criterion is that items should be **novel**, since the goal of recommendation closely tied to discovery [39]; we want to avoid recommending highly popular items that often appear on the platform's front page. Finally, the collection of recommended items across all requests should have high **coverage**, ensuring a diverse range of recommendations. This breadth of visibility is especially crucial for the success of a platform that relies on millions of user-generated content.

3 Methods

3.1 Dataset

3.1.1 Requests. We identify a Reddit community /r/Roblox, where users discuss a wide range of topics about Roblox and its games. Here, we find that some posts are asking for Roblox game recommendations. We sample the posts by using the Python Reddit



```
Games to play with my
8-year old son?
He likes cars and
building stuff. We
liked the build a boat
game but we are open
to anything else
except for horror
games. Preferably
something on Xbox.
```

```
formatted_intent =

{
   "like": {
     "games": ["Build a Boat"],
     "properties": ["cars", "building"],
     "devices": ["CONSOLE"]
   },
   "dislike": {
     "genres": ["horror"]
   },
   "demographics": {
     "ages": ["0-8"]
   }
}
```

```
execute_tools(formatted_intent)
"""
Selects and executes tools according to policy P and returns the
results into a readable format.
"""

Examples of tools:
• get_game_id_from_fuzzy_name("Build a Boat") → 210851291
• get_game_description(210851291) → "This game involves…"
• get_similar_games_cf(210851291) → [27958020, …]
• get_search_results("cars") → [705059969, 274816972, …]
• get_game_genre(4623386862) → "Horror"
• get_games_by_age_group("8-12") → [4777817887, …]
• is_device_compatible(705059969, "CONSOLE") → True
• ...
```

```
Lookup 'Build A Boat For Treasure':
Genre: Sandbox. This game involves designing…

Users who played 'Build A Boat For Treasure' also played:
1. The Strongest Battlegrounds - Genre: Fighting. Train and…

Search results for 'cars':
1. World Of Cars - Genre: All. In this game, players…
```

```
1. Build Together
2. Car Crushers 2
3. Building Blocks Simulator
4. Stepford County Railway
5. ...
```

Figure 3: Overview of OMuleT. Orange boxes are in the user interface (a user inputs a raw request and observes recommended items); blue boxes are where LLMs are used; green boxes are where tools are used.

API Wrapper (PRAW),[1] using keyphrases such as 'recommend me games' and 'what games to play'. We further filter the posts by asking GPT-3.5 [28] to judge whether the request is asking for game recommendations, and then removing the ones that are not. This process may still leave a handful of irrelevant posts, such as a game developer asking for recommendations on what game to make. As such, we manually remove the remaining irrelevant posts (73 out of 629 posts). We denote the resulting 556 posts as *requests*.

3.1.2 Human Oracles. For each request, there are comments from other users in the community that recommend games that are relevant to the request. We regard these games as *human oracles*. When users mention game names, they may not precisely state the exact name, such as referring it with an acronym (e.g., 'MM2' instead of 'Murder Mystery 2') or dropping out parts of the name (e.g., 'Bloxburg' instead of 'Welcome to Bloxburg'). To handle this, we ask GPT-3.5 to extract any phrases that might be a game name (to ensure high recall), and link it to real game IDs using the Roblox search API[2] (to ensure high precision).[3] To ensure the quality of oracles, we measure community agreement through the net upvotes of comments. For each request, we keep games that have at least one net upvote and discard the rest. We obtain 553 requests with at least one oracle. There are 14.21(±32.22) oracles per request and 2074 unique games in total.

3.1.3 From Oracles to Ground-Truth Items. Human oracles may be noisy (i.e., some games are irrelevant) or insufficient (i.e., there may be more games that are relevant to a request). We refine the set of recommendations through a two-step process. First, for each request, we generate a candidate set of games. This is done by using the oracles: we obtain games similar to the oracle by using two Roblox APIs.[4] Oracles and similar games are added to the candidate set by prioritizing their frequency across all oracles and APIs, with up to 30 candidates generated per request. Second, human experts determine whether each candidate is relevant to the request. These experts are highly knowledgeable about Roblox games, but if they are unfamiliar with a displayed game, they must play it to evaluate its relevance. Additionally, to ensure safety, experts are instructed to remove any age-inappropriate games based on the request. We denote the resulting games as the *ground-truth items* for a given request. Due to resource constraints, 208 requests were processed using the above method. Each request has an average of 9.06(±9.03) ground-truth games, totaling 1031 unique games.

3.2 Proposed Framework: OMuleT

3.2.1 System Overview. Our system overview is depicted in Figure 3. When a user submits a request, an LLM generates a dictionary summarizing the user's preference, denoted as the **formatted intent**. The formatted intent is given as input to the **tool execution policy**, which selects the tools and arguments to execute and returns an **execution output** in natural language. Note that the execution output is not the final recommendation; it contains the relevant information that would augment the LLM with external

[1]https://praw.readthedocs.io/en/stable/
[2]Precisely, we use one of our tools, `get_game_id_from_fuzzy_name` (see Table 1).
[3]To illustrate, 'weirdest game on roblox' is a game name; 'fun surfing' is not.
[4]We use tools: `get_similar_games_cf` and `get_similar_games_content` (Table 2).

Table 1: List of tools in our current toolbox. More tools may be added to our framework.

Group	Tool	Input	Output	Description
Lookup	`get_game_name`	Game ID	Game name	Return the game name.
	`get_game_genre`		Game genre	Return the game genre among the 21 predefined categories, e.g., 'RPG'.
	`get_game_description`		Game description	Return a 2-3 sentence summary of what the game is about and how it is played.
	`get_game_rank`		Game rank	Return the game rank by number of upvotes.
	`is_device_compatible`	Game ID, Device	True or False	Determine if the game is compatible with the given device, e.g., 'CONSOLE'.
Linking	`get_game_id_from_fuzzy_name`	Fuzzy name	Game ID	Given an approximate game name, return a game ID that is highly likely to correspond to this game. If no game is found, return nothing. E.g., 'MM2' $\longrightarrow$ ID for 'Murder Mystery 2'
	`fuzzy_genre_to_genres`	Fuzzy genre	Genres	Given a fuzzy genre name, return a list of predefined genres that are likely to correspond to this genre. If no genre is found, return nothing. E.g., 'simulation' $\longrightarrow$ ['Simulator/Clicker', 'Tycoon/Management Sim']
Retrieval	`get_search_results`	Simple query	Game IDs	Use the search API to return the games relevant to a simple query (maximum 3 words).
	`get_similar_games_cf`	Game ID		Use the collaborative filtering API to return 'users who played this game also played $\cdots$.'
	`get_similar_games_content`	Game ID		Use the SBERT [32] embeddings to return the games that have similar descriptions.
	`get_games_by_age_group`	Age group		Get games commonly played among the given age group, e.g., '18-24'.
	`get_default_games`	# games		Randomly sample games from the top 100 games. May be needed when a user request is too generic.
Formatting	`get_game_info_str`	Game ID	Formatted info.	Return a string of game information in the following format: '{game name} – {genre}. {description}'
	`game_ids_to_enum_game_info`	Game IDs	Formatted info.	Return a string of enumerated game information in the order of the given list.

knowledge (e.g., item information) so that it generates better recommendations. In the recommendation phase, both the raw request and execution output are provided to the LLM, which generates a list of game names. Each game is then linked to a real item in the Roblox database and displayed to the user.

3.2.2 *Formatted Intent Generation.*

While it is possible to make LLMs directly generate code policies for tool execution [23, 41], we later show that this approach is not effective for our task (Section 4). Furthermore, from an industry perspective, we want the system to be *transparent* (we can see how the system is operating), and *controllable* (we can easily control and fix how the system works). In this sense, we propose the following design: let the LLM first process the raw request into a formatted intent D_{int}, and and execute a handcrafted policy P based on the formatted intent. This design has several practical benefits: (1) it allows us to view the intermediate stage (formatted intent), helping us assess incoming requests and verify whether they are understood or parsed correctly; (2) instead of depending on LLMs for code generation—which can be a black box and have syntax errors—we rely on human experts for a better understanding and execution of tools; (3) it yields better performance than using LLM-generated policies. Specifically, we use the following prompt:

"Given a user's recommendation request, format the user's preference into a JSON format. Fill in the following template of dict[str, dict[str, list]] with the relevant information accurately extracted from the user's request: <template> <demonstrations>".

The <template> consists of preferences and user demographics, where each preference ('like' and 'dislike') contains four fields:

- Genres: approximate game genres that do not need to match Roblox's official categories exactly
- Game names: approximate game names that do not need to match Roblox's game names exactly
- Properties: simple keyphrases describing the features or elements of a game

- Devices: a subset of ‘DESKTOP’, ‘PHONE’, ‘TABLET’, ‘CONSOLE’, and ‘VR’.

User demographics are composed of two fields:

- Ages: age group(s) of user(s) from a subset of ‘0-8’, ‘9-12’, ‘13-17’, ‘18-24’, ‘25-34’, ‘35plus’.[5]
- Genders: gender(s) of user(s) inferred from explicit information in the request (e.g., ‘my son’ $\rightarrow$ ‘MALE’).[6]

We provide 5 demonstrations, which, compared to no demonstration, results in a more stable generation of formatted intents with the proper template and syntax.

3.2.3 Toolbox. Our tools are Python functions, each performing a specific retrieval task that is potentially useful for recommendation. We present the entire list of tools in Table 1, along with each tool’s input, output, and description. Tools are broadly classified into four categories: **Lookup** tools return simple game metadata from the Roblox database. Lookup tools can be used for informing LLMs with item knowledge (e.g., game descriptions), or filtering items based on attributes (e.g., compatible devices). Although some works propose to employ a single lookup tool by SQL query generation [44, 45], this method may not be suitable in many applications, including ours. For example, the database used in production may not be in a structure where LLMs can generate accurate and efficient SQL queries. Instead, we propose to have multiple simple tools for accessing the database. Such design is also important to making the system transparent and controllable. **Linking** tools match game names and genres from user utterances to corresponding entities in the Roblox database. These tools are essential for handling real user requests where exact game IDs or genre categories are not used. Although implementing a drop-down list in the user interface [22] could bypass this issue, we believe it reduces engagement by requiring users to select from a list instead of typing naturally. **Retrieval** tools retrieve games that may be relevant to the user’s request. For example, if a user references a game to express their preference, the similarity-search tools retrieve similar games using collaborative filtering (based on similar users) and game content (based on descriptions). While similar to candidate generators, recommendations are not necessarily confined to the retrieved games. Instead, the purpose of retrieval tools is to make LLMs be ‘aware’ of the diverse items in the system instead of generating the most popular ones. Later we show that the absence of these tools results in much lesser diversity of recommended items. **Formatting** tools summarize the tool execution results into a natural language format, which would be provided in the prompt for the recommendation stage.

Note that we do not use ranking tools. Instead of using a ranking tool to output the final recommendations [13], we let LLMs do the eventual recommendation by having them enumerate a list of items, as we later discuss in Section 3.2.5.

[5]We use LLMs to generate devices and age groups from predefined categories since there are only a handful of them and doing so does not require domain knowledge. Entity linking of genres and game names require specialized tools.

[6]After collecting the formatted intents, we notice that genders are rarely mentioned, so we have not created a relevant tool for gender. If a model is deployed within a production platform, incorporating demographics from user account profiles is a feasible enhancement we may consider in future work.

Algorithm 1 Tool execution policy P

```
1: Input: Formatted intent dictionary $D_{int}$
2: Output: Results dictionary $D_{aug}$
3: Initialize: $D_{aug} \leftarrow \{\}$
4: for game in $D_{int}$[liked games] do
5:     $D_{aug} \leftarrow D_{aug} \cup$ lookup(game)                          ▷ Lookup
6:     $D_{aug} \leftarrow D_{aug} \cup$ similar(game)                         ▷ Similar
7: $D_{aug} \leftarrow D_{aug} \cup$ search($D_{int}$[liked genres])             ▷ Search
8: if $D_{aug} = \{\}$ then
9:     $D_{aug} \leftarrow D_{aug} \cup$ search($D_{int}$[liked properties])     ▷ Search
10: for game in $D_{int}$[disliked games] do
11:     $D_{aug} \leftarrow D_{aug} \cup$ lookup(game)                         ▷ Lookup
12: $D_{aug} \leftarrow D_{aug} \cup$ games_by_age($D_{int}$[user age groups])    ▷ Age
13: if $D_{aug} = \{\}$ then
14:     $D_{aug} \leftarrow$ default_games(30)    ▷ If the user’s request is too
    generic, i.e., $D_{aug}$ is empty so far, randomly sample 30 games
    from top-100 games.
15: for game in $D_{aug}$[similar, search, age results] do                 ▷ Filter
16:     if genre(game) in $D_{int}$[disliked genres] then
17:         $D_{aug} \leftarrow D_{aug} \setminus$ game
18:     if incompatible(game, $D_{int}$[preferred devices]) then
19:         $D_{aug} \leftarrow D_{aug} \setminus$ game
20: $D_{aug} \leftarrow$ format($D_{aug}$)                                      ▷ Format
21: return $D_{aug}$
```

3.2.4 Tool Execution. We describe our tool execution policy P: $D_{int} \rightarrow D_{aug}$ in Algorithm 1.[7] The policy goes through each (key, value) in the formatted intent and runs the corresponding tools, adding information to D_{aug} that would potentially be helpful to the recommendation stage. Then the policy goes through D_{aug} again and filters items that can be sources of noise (e.g., games that are incompatible with the user’s preferred devices). While we can skip the filtering and let the LLM disregard irrelevant items in the final recommendation stage, we find that simply filtering items in advance improves recommendation performance. Finally, the policy uses the formatting tools to convert D_{aug} into a readable format to be passed into the recommendation stage. For example,

{‘Users who played id0 also played’: [id1, id2, $\cdots$]}

becomes

Users who played ‘Da Amazing Bunker Simulator’ also played:
1. RetroStudio — Genre: Sandbox. This game allows players to create $\cdots$

3.2.5 Recommendation. To generate high-quality recommendations, a model needs to accurately understand complex and nuanced requests. LLMs excel in natural language understanding to such an extent that they surpass traditional, smaller models at conversational recommendation [11]. As such, instead of having a separate tool (e.g., for ranking) to generate the final recommendations, we prompt an LLM with the raw request, tool execution output D_{aug}, and an instruction to generate a list of relevant items. This method

[7]For presentation simplicity, we omit linking tools and abbreviate tool names.

utilizes the LLM's language capability (i.e., understanding raw request) and augments its weakness by providing external knowledge (i.e., tool execution output). We use the following instruction: *'Given the following request, provide recommendations. Enumerate 20 Roblox game names (1., 2., ...) in the order of relevance. Don't say anything else.'* We augment the LLM with D_{aug} by adding: *'Using the above information along with your own knowledge and reasoning, provide the best recommendations that fulfill the request.'*

4 Experiments

4.1 Evaluation Metrics

We use multiple evaluation metrics for relevance, novelty, and coverage of recommended items. Additionally, since we are using LLMs as recommenders, we measure factuality to understand whether models are hallucinating.

4.1.1 Relevance. **Hit@k** evaluates whether a ground-truth item is included in the top-k recommendations. **Precision@k** is the proportion of ground-truth items in the recommendations. **Similar@k** is the similarity of ground-truth items and recommended items by computing the cosine distance between the embedding centroids. In our work, we use SimCSE [5] embeddings obtained from item descriptions. We average each of the metrics across all requests.

4.1.2 Novelty. The concept of novelty in recommender systems can vary, but it is commonly linked to an item's popularity, such as the number of ratings it has received [14, 39]. In a similar vein, we use a metric that uses item popularity, where **Pop50@k** is the proportion of items in the top 50 most popular (or well-known) games, ranked by upvotes. Lower values are better since popular items are often listed on the Roblox front page, and our objective is to help users discover unfamiliar items. We also use **RPop50@k**, which computes the ratio of Pop50@k for the recommended items to that of the ground-truth items. Closer value to 1 indicates that the recommendations are as novel as the ground-truth items.

4.1.3 Coverage. **Entropy@k** measures the diversity of recommended items across all requests, formally computed by the following equation: Entropy@k $= -\sum_i p_i \log(p_i)$, where p_i is defined as the frequency of item i across the top-k recommendations. Higher entropy indicates a wider coverage of items [14, 30]. **MaxFreq@k** identifies the most frequently recommended item, and computes the proportion of requests that this item is recommended. For example, if 'Adopt Me!' appears in the top-10 list in 60% of the requests, then MaxFreq@10 is 0.60. A lower value is preferable, as it indicates that the system avoids recommending the same item repeatedly.

4.1.4 Factuality. **Factual@k** measures the proportion of real items in the top-k list. If the tool `get_id_from_fuzzy_name` returns nothing, we regard the game name as hallucinated. While factuality can be easily addressed by displaying only the actual items to the user, it remains an important metric for understanding model performance. We compute other metrics after filtering out hallucinated items.

4.2 Setup

4.2.1 LLMs. We use LLaMA-405B [1] and GPT-4o [29]. The temperatures of LLMs are set to 0 for deterministic results.[8] For simplicity, we use the same LLMs for formatting and recommendation. In practice, the two stages can be run by different LLMs.

4.2.2 Baselines. Previous works (see Section 6) use zero-shot [11, 33, 52] or tool-augmented [13, 16, 19, 44, 47] LLMs for CRS. Since the tools in each paper are often domain-specific and difficult to apply in our work, we perform ablation tests to show the necessity of a large number of tools, which distinguishes us from existing methods. RAG-based approaches that retrieve similar queries from the training corpus [48] are also unsuitable for our setting due to a small volume of available queries (which we entirely use for evaluation). While some works fine-tune LLMs with traditional user-item data or synthetic queries [15, 54], we do not consider them as baselines since we want to incorporate external knowledge without the cost of fine-tuning. As such, our baselines are as follows:

- **Pop** randomly selects k items from the top-50 list.
- **Base LLM** is an LLM without any tool augmentations.
- **Base LLM + Div** is a slight variant that encourages the base LLM to generate lesser-known items by simply adding the following instruction: *'The games should be diverse and not too well-known (should be new to the user).'*
- **OMuLeT w/** P_{LLM} replaces the handcrafted policy P with LLM-generated ones, to observe whether LLMs can generate better policies than P, as previous works suggest [13, 23]. We provide the LLM with the raw request, formatted intent, and a list of available tools and instruct to generate a code in Python that outputs D_{aug}.[9]

4.3 Results

We organize the results into multiple research questions. Results for Q1-3 are in Table 2, and the ablation study for Q4 is in Figure 4.

Q1. Is OMuLeT more effective than base LLMs? OMuLeT outperforms base LLMs in all metrics for the human-annotated dataset (see Table 2). For the full dataset, OMuLeT outperforms base LLaMA-405B in all metrics and GPT-4o in all but Hit and Precision; this discrepancy could be attributed to the lack of accurate ground-truth items for the full dataset. Base LLMs have particularly poor novelty and coverage; LLaMA-405B recommends top-50 items ×3.19 more frequently than the ground-truths, and recommends the most frequent item ('Natural Disaster Survival') in 43% of requests. This is in contrast to OMuLeT, where LLaMA-405B recommends top-50 items only ×1.31 more than the ground-truths, and recommends the most frequent item in 10% of requests. While OMuLeT achieves near-perfect factuality (> 99%), base LLMs generate hallucinations among 21% (LLaMA-405B) and 11% (GPT-4o) of top-10 recommendations.

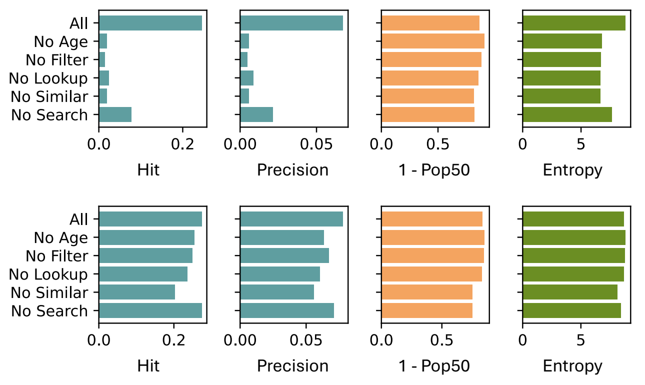
Q2. Is fixed P better than LLM-generated policies? We experiment to see if LLMs can generate their own policies, P_{LLM}, per request using the same toolbox, to determine if they can create more effective, customized policies. We find that although LLMs generate reasonable policies, relevance metrics significantly drop

[8]We also tried other temperature values, but the differences were insignificant.
[9]We also tried providing a demonstration of a handcrafted policy, but we observe that this makes LLM replicate the handcrafted policy.

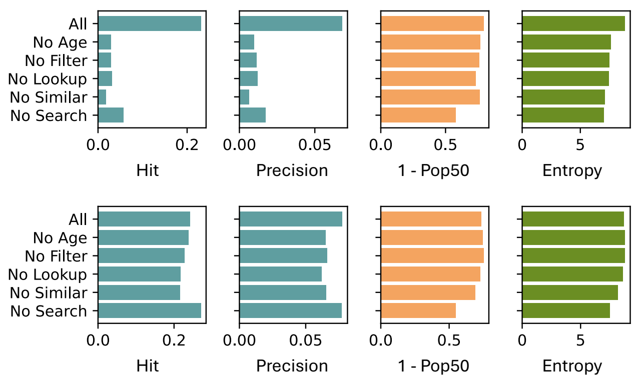
	Method	Factuality	Relevance			Novelty		Coverage	
		Factual (↑)	Hit (↑)	Precision (↑)	Sim (↑)	Pop50 (↓)	RPop50 (↓)	Entropy (↑)	MaxFreq (↓)
	Pop	1.00 1.00	.08 .14	.02.04	.91 .89	1.00 1.00	10.31 7.97	5.61 5.64	0.15 .12
LLaMA-405B	Base LLM	.84 .88	.22 **.23**	.06 .06	.91 .88	.35 .48	3.60 3.84	7.16 6.57	.36 .53
	Base LLM + Div	.68 .70	.13 .16	.03 .04	.86 .84	.15 **.17**	1.52 1.39	7.66 7.63	.18 .27
	OMuleT w/ P_{LLM}	.98 .98	.22 18	.05 .05	.92 **.89**	.14 **.17**	1.39 **1.35**	**8.97** **9.18**	.10 **.12**
	OMuleT w/ P	**1.00** **.99**	**.25** **.23**	**.07** **.07**	**.93** **.89**	**.13** .21	**1.38** 1.63	8.81 8.85	**.05** .16
GPT-4o	Base LLM	.90 .94	.26 **.29**	.07 **.09**	.90 .88	.42 .56	4.34 4.48	7.17 6.64	.20 .39
	Base LLM + Div	.59 .64	.16 .18	.04 .05	.73 .73	**.11** **.12**	**1.10** **.96**	8.15 8.53	**.07** **.10**
	OMuleT w/ P_{LLM}	.98 **.99**	.22 .19	.06 .06	**.93** **.90**	.16 .21	1.60 1.67	**8.73** **8.97**	.10 **.10**
	OMuleT w/ P	**.99** **.99**	**.27** .24	**.08** .08	**.93** .89	.17 .27	1.71 2.14	8.68 8.71	**.07** .12

	Method	Factuality	Relevance			Novelty		Coverage	
		Factual (↑)	Hit (↑)	Precise (↑)	Sim (↑)	Pop50 (↓)	RPop50 (↓)	Entropy (↑)	MaxFreq (↓)
	Pop	1.00 1.00	.15 .19	.02 .03	.93 .90	1.00 1.00	11.23 8.40	5.63 5.64	.26 .24
LLaMA-405B	Base LLM	.79 .83	.25 .28	.04 .05	.92 .89	.28 .40	3.19 3.32	7.68 7.26	.43 .60
	Base LLM + Div	.64 .67	.16 .19	.02 .03	.87 .85	**.11** .14	**1.25** **.21**	7.82 7.79	.17 .30
	OMuleT w/ P_{LLM}	.98 .98	.30 .24	.04 .04	.93 .90	.13 **.17**	1.50 1.43	**9.50** **9.60**	.16 .23
	OMuleT w/ P	**1.00** **.99**	**.36** **.31**	**.05** **.06**	**.94** **.91**	.12 .19	1.31 1.63	9.48 9.43	**.10** **.19**
GPT-4o	Base LLM	.89.93	.36 **.38**	**.06** **.08**	.92 .89	.38 .53	4.24 4.43	7.62 7.08	.28 .50
	Base LLM + Div	.57.61	.20 .23	.03 .04	.74 .74	**.09** **.12**	**1.04** **.97**	8.37 8.63	**.08** **.14**
	OMuleT w/ P_{LLM}	.98 **.99**	.33 .28	.05 .05	**.94** **.91**	.15 .22	1.72 1.84	9.23 **9.33**	.13 .15
	OMuleT w/ P	**.99**.99	**.38** .33	**.06** .06	**.94** **.91**	.14 .25	1.61 2.13	**9.31** 9.21	.12 .24

Table 2: Results for top 5 (above) and 10 (below) recommendations, on human-annotated and full (colored grey) datasets.



(a) Annotated dataset: LLaMA-405B (above) GPT-4o (below)



(b) Full dataset: LLaMA-405B (above) GPT-4o (below)

Figure 4: Ablation study. Recommendations are more relevant if we use more tools. One exception is when removing the search tool for GPT-4o: relevance increases, but this comes at a relatively large cost to both novelty (1−Pop50) and diversity (Entropy). Above are results for $k = 5$ and we observe similar trends for different k values.

compared to simply using the fixed policy P. We observe high coverage (Entropy) in some cases, but the overall results show that there is little or no advantage using P_{LLM} over fixed P, especially considering that the former approach is less transparent and controllable. That said, OMuleT with P_{LLM} consistently outperforms base LLMs in factuality, novelty, and coverage, and occasionally in relevance, suggesting that retrieving any relevant results is preferable to none for factual and diverse recommendations.

Q3. Can we prompt LLMs to recommend more diverse items? Base LLMs indeed generate more diverse items (higher novelty and coverage) when explicitly prompted to do so, but this leads to a significant loss in relevance (55-64% of unprompted) and factuality (64-81% of unprompted). While simple prompting yields even higher novelty than OMuleT when $k = 10$, the differences are relatively small (1.25 v.s. 1.31 for LLaMA and 1.04 vs. 1.61 for GPT-4o in RPop).

Q4. Do we need all the tools? Figure 4 shows the results of our ablation study, where we remove each tool to observe the impact on performance. We find that using all the tools generally improves relevance, with two unexpected results. One is that the performance of LLaMA-405o significantly drops when any tool is omitted. A possible explanation is that augmenting with partial information may mislead the model (e.g., by providing similar games but not age-relevant games). The model may also be sensitive to noise when the filtering tool is not used. Another interesting result is that dropping the search tool can slightly increase relevance (although at the notable cost of novelty and converage) for GPT-4o. To understand this, we examined the search tool's outputs. One issue is that the Roblox search API sometimes returns noisy results, such as retrieving low-quality games. But a more fundamental problem is that many user-described properties, such as 'sweet', 'not too horror', 'no progression', 'nice people', and 'unique premise', can be ambiguous or incompatible with search queries. OMuleT is intended to handle such nuanced requests by letting LLMs understand the request holistically (e.g., 'sweet' as the game 'Oobja', or 'not too horror' as less intense than 'The Mimic') and use the provided game descriptions to match them with the request. However, their descriptions alone may not provide enough context to accurately match games with requests.[10] One way to address this issue is to obtain descriptions of actual gameplay or user opinion, which we consider as future improvements. In terms of novelty and coverage, using all tools yields similar or better results than omitting any.

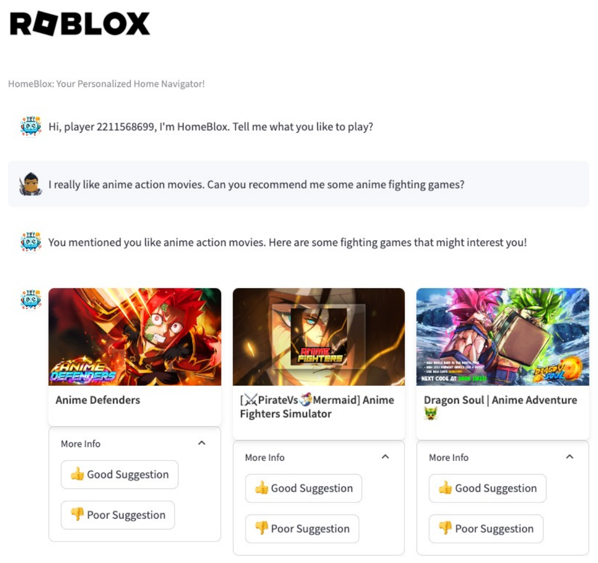
5 Deployment

To perform a feasibility study and identify the best implementation practices, we launch an internally hosted chatbot (see Figure 5). Our application is built on a full-stack server using Streamlit [37], which simplifies creating an interactive UI and managing backend operations. We deploy the application in an internal datacenter using HashiCorp's Nomad and Consul [9] for cluster orchestration, deployment, and configuration. Several key areas are under evaluation to assess the feasibility of transitioning the chatbot to production. First is ensuring system safety by preventing irrelevant queries, policy violations, and jailbreak attempts (see Section 8). Second is latency and scalability. Our current chatbot takes several seconds per query to generate results. Further studies are necessary to understand user tolerance for latency and explore techniques to enhance inference efficiency at scale.

6 Related Work

Conversational Recommender Systems. There are two categories of works based on the evaluation approach: interactive and dataset-based. In interactive evaluation, a user simulator replaces real users, and the problem is often framed into item or attribute selection for preference elicitation [3, 18, 38, 55]. In this approach, user simulators may fall short of reflecting real users [51]. Dataset-based evaluation recommends items given prior utterance. Most existing datasets are crowd-sourced [10, 22, 26], where workers role-play as seeker and recommender. Early works propose using two separate modules, language understanding and recommendation, while more recent work suggests merging the two [43]. Most recently, zero-shot LLMs have shown to outperform all previous methods, especially for complex user utterances [11].

[10]For example, in the case of 'The Mimic,' the description mentions it is a horror game with jumpscares, but it does not convey the intensity of the horror compared to other games. Such information could be gleaned from user opinion data, such as reviews.


HomeBlox: Your Personalized Home Navigator!

Hi, player 2211568699, I'm HomeBlox. Tell me what you like to play?

I really like anime action movies. Can you recommend me some anime fighting games?

You mentioned you like anime action movies. Here are some fighting games that might interest you!

Anime Defenders

[⚔️PirateVs 🧜Mermaid] Anime Fighters Simulator

Dragon Soul | Anime Adventure

Figure 5: Screenshot of the deployed UI. We add simple greeting and explanations for a more natural conversation, and thumbs up and down buttons for obtaining feedback.

LLMs for Recommendation. Since LLMs take language inputs, most recommendation task that uses an LLM inherently becomes 'conversational'. Often, queries are generated by inserting non-CRS datasets (e.g., user-item interactions) into templates [6, 8, 12, 15, 20, 42]. LLMs can be fine-tuned with such queries [27, 54], and further enhanced by incorporating collaborative filtering information during training [17, 49, 57, 59]. In contrast, our focus is on the user requests expressed in their own words, not bound in templates.

Tool-Augmented LLMs. Recent works explore using LLMs to create agents that can perform complex interactive tasks. Applications include robotic control [2], scientific reasoning [24], and question answering [36, 50]. Solving such tasks often requires using tools [21, 31]. Tools are functions external to the LLM [45], and can help agents access external knowledge bases [7, 50], perform arithmetic operations [7, 34], use specialized models [25, 35], and interact with the world [40, 58]. Agents can even create simple tools and add them to the toolbox [46, 53]. Some works explore the possibility of having the agent generate a policy for using tools [23, 41], but we have shown in our experiments that using a fixed policy is more effective for our task.

Tool-Augmented LLMs for Recommendation. In recommendation, access to external knowledge is crucial because items are frequently added or removed, their information is updated (e.g., content updates or shifts in popularity), and external factors (e.g., seasonal demand) can influence user preference. As such, recent works propose tool-augmented LLMs for recommendation [13, 16, 19, 44, 47] to retrieve relevant information from an external knowledge base. From a practical perspective, we are faced with several limitations in directly applying this work: queries are often synthetic [13, 16, 44], which are different from real users; tools used in previous work are often unavailable in some use cases including ours, e.g., review-based item retrieval [16]. While demonstration papers [4, 16, 19] focus on implementing working systems, our work complements these efforts by providing extensive evaluation.

7 Conclusion

This work aims to advance practical conversational recommender systems by collecting a dataset of real user requests and proposing a novel approach to augmenting large language models with multiple tools. Our study includes comprehensive experiments and deployment insights. One limitation is we focus on game recommendations, which may not generalize to other domains. Additionally, the Reddit dataset may not fully represent all user types. As future work, we plan to develop models based on larger datasets.

8 Ethical Considerations

In developing our system, we prioritize ethical considerations, particularly in the areas of fairness, diversity, and system integrity. To address fairness and diversity, the evaluation of our system is designed with native support for these principles, using carefully processed datasets and beyond-accuracy metrics to ensure equitable recommendations across users and items. In terms of integrity, we implement dedicated modules to handle the following:

- Jailbreak prevention: A mechanism to protect against external manipulation and unauthorized system exploitation.
- Integrity verification: A mechanism to ensure the safety of recommendations and the words used by the conversational system, ensuring reliable outputs.

While these measures significantly reduce the risks associated with fairness, diversity, and integrity, it is important to acknowledge that due to the inherent complexity of large language models, these issues cannot be entirely eliminated. The field is rapidly evolving, and ongoing research is essential to further refine and enhance these protections. In summary, our system incorporates robust solutions to address ethical concerns, though we recognize the need for continuous improvement as part of the broader research landscape.

References

[1] Meta AI. 2024. LLaMA 3.1: 405B Parameter Model. https://ai.meta.com/llama. Accessed: 2024-08-09.

[2] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. 2023. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on robot learning*. PMLR.

[3] Konstantina Christakopoulou, Filip Radlinski, and Katja Hofmann. 2016. Towards conversational recommender systems. In *KDD*.

[4] Luke Friedman, Sameer Ahuja, David Allen, Zhenning Tan, Hakim Sidahmed, Changbo Long, Jun Xie, Gabriel Schubiner, Ajay Patel, Harsh Lara, et al. 2023. Leveraging large language models in conversational recommender systems. *arXiv preprint arXiv:2305.07961* (2023).

[5] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821* (2021).

[6] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. 2022. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *RecSys*.

[7] Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2024. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. In *NeurIPS*.

[8] Jesse Harte, Wouter Zorgdrager, Panos Louridas, Asterios Katsifodimos, Dietmar Jannach, and Marios Fragkoulis. 2023. Leveraging large language models for sequential recommendation. In *RecSys*.

[9] Hashicorp. 2024. Nomad and Consul. https://developer.hashicorp.com/consul/docs/connect/nomad. Accessed: 2024-08-09.

[10] Shirley Anugrah Hayati, Dongyeop Kang, Qingxiaoyang Zhu, Weiyan Shi, and Zhou Yu. 2020. Inspired: Toward sociable recommendation dialog systems. *arXiv preprint arXiv:2009.14306* (2020).

[11] Zhankui He, Zhouhang Xie, Rahul Jha, Harald Steck, Dawen Liang, Yesu Feng, Bodhisattwa Prasad Majumder, Nathan Kallus, and Julian McAuley. 2023. Large language models as zero-shot conversational recommenders. In *CIKM*.

[12] Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. 2024. Large language models are zero-shot rankers for recommender systems. In *ECIR*.

[13] Xu Huang, Jianxun Lian, Yuxuan Lei, Jing Yao, Defu Lian, and Xing Xie. 2023. Recommender ai agent: Integrating large language models for interactive recommendations. *arXiv preprint arXiv:2308.16505* (2023).

[14] Marius Kaminskas and Derek Bridge. 2016. Diversity, serendipity, novelty, and coverage: a survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM TiiS* 7, 1 (2016), 1–42.

[15] 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).

[16] Sara Kemper, Justin Cui, Kai Dicarlantonio, Kathy Lin, Danjie Tang, Anton Korikov, and Scott Sanner. 2024. Retrieval-Augmented Conversational Recommendation with Prompt-based Semi-Structured Natural Language State Tracking. In *SIGIR*.

[17] Sein Kim, Hongseok Kang, Seungyoon Choi, Donghyun Kim, Minchul Yang, and Chanyoung Park. 2024. Large Language Models meet Collaborative Filtering: An Efficient All-round LLM-based Recommender System. *arXiv preprint arXiv:2404.11343* (2024).

[18] Wenqiang Lei, Xiangnan He, Yisong Miao, Qingyun Wu, Richang Hong, Min-Yen Kan, and Tat-Seng Chua. 2020. Estimation-action-reflection: Towards deep interaction between conversational and recommender systems. In *WSDM*.

[19] Chuang Li, Yang Deng, Hengchang Hu, Min-Yen Kan, and Haizhou Li. 2024. Incorporating External Knowledge and Goal Guidance for LLM-based Conversational Recommender Systems. *arXiv preprint arXiv:2405.01868* (2024).

[20] Lei Li, Yongfeng Zhang, and Li Chen. 2023. Prompt distillation for efficient llm-based recommendation. In *CIKM*.

[21] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 3102–3116. https://doi.org/10.18653/v1/2023.emnlp-main.187

[22] Raymond Li, Samira Ebrahimi Kahou, Hannes Schulz, Vincent Michalski, Laurent Charlin, and Chris Pal. 2018. Towards deep conversational recommendations. In *NeurIPS*.

[23] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. In *ICRA*.

[24] Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. 2024. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. In *NeurIPS*.

[25] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. In *NeurIPS*.

[26] Seungwhan Moon, Pararth Shah, Anuj Kumar, and Rajen Subba. 2019. Opendialkg: Explainable conversational reasoning with attention-based walks over knowledge graphs. In *ACL*.

[27] Sheshera Mysore, Andrew McCallum, and Hamed Zamani. 2023. Large language model augmented narrative driven recommendations. In *RecSys*.

[28] OpenAI. 2024. GPT-3.5 Turbo. https://platform.openai.com/docs/models/gpt-3-5-turbo. Accessed: 2024-08-09.

[29] OpenAI. 2024. GPT-4o. https://platform.openai.com/docs/models/gpt-4o. Accessed: 2024-08-09.

[30] Lijing Qin and Xiaoyan Zhu. 2013. Promoting diversity in recommendation by entropy regularizer. In *IJCAI*.

[31] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789* (2023).

[32] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[33] Scott Sanner, Krisztian Balog, Filip Radlinski, Ben Wedin, and Lucas Dixon. 2023. Large language models are competitive near cold-start recommenders for language-and item-based preferences. In *RecSys*.

[34] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*.

[35] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. In *NeurIPS*.

[36] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*.

[37] Streamlit. 2024. Streamlit. https://streamlit.io/. Accessed: 2024-08-09.

[38] Yueming Sun and Yi Zhang. 2018. Conversational recommender system. In *SIGIR*.

[39] Saúl Vargas and Pablo Castells. 2011. Rank and relevance in novelty and diversity metrics for recommender systems. In *RecSys*.

[40] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291* (2023).

[41] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030* (2024).

[42] Xinfeng Wang, Jin Cui, Yoshimi Suzuki, and Fumiyo Fukumoto. 2024. RDRec: Rationale Distillation for LLM-based Recommendation. *arXiv preprint arXiv:2405.10587* (2024).

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

[44] Yancheng Wang, Ziyan Jiang, Zheng Chen, Fan Yang, Yingxue Zhou, Eunah Cho, Xing Fan, Yanbin Lu, Xiaojiang Huang, and Yingzhen Yang. 2024. RecMind: Large Language Model Powered Agent For Recommendation. In *NAACL (Findings)*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.).

[45] Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. 2024. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452* (2024).

[46] Zhiruo Wang, Daniel Fried, and Graham Neubig. 2024. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks. *arXiv preprint arXiv:2401.12869* (2024).

[47] Yunjia Xi, Weiwen Liu, Jianghao Lin, Bo Chen, Ruiming Tang, Weinan Zhang, and Yong Yu. 2024. MemoCRS: Memory-enhanced Sequential Conversational Recommender Systems with Large Language Models. *arXiv preprint arXiv:2407.04960* (2024).

[48] Zhouhang Xie, Junda Wu, Hyunsik Jeon, Zhankui He, Harald Steck, Rahul Jha, Dawen Liang, Nathan Kallus, and Julian McAuley. 2024. Neighborhood-Based Collaborative Filtering for Conversational Recommendation. In *RecSys*.

[49] Li Yang, Anushya Subbiah, Hardik Patel, Judith Yue Li, Yanwei Song, Reza Mirghaderi, and Vikram Aggarwal. 2024. Item-Language Model for Conversational Recommendation. *arXiv preprint arXiv:2406.02844* (2024).

[50] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).

[51] Se-eun Yoon, Zhankui He, Jessica Echterhoff, and Julian McAuley. 2024. Evaluating Large Language Models as Generative User Simulators for Conversational Recommendation. In *NACCL*.

[52] Se-eun Yoon, Hyunsik Jeon, and Julian McAuley. 2024. Imagery as Inquiry: Exploring A Multimodal Dataset for Conversational Recommendation. *arXiv preprint arXiv:2405.14142* (2024).

[53] Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R Fung, Hao Peng, and Heng Ji. 2023. Craft: Customizing llms by creating and retrieving from specialized toolsets. *arXiv preprint arXiv:2309.17428* (2023).

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

[55] Xiaoying Zhang, Hong Xie, Hang Li, and John CS Lui. 2020. Conversational contextual bandit: Algorithm and application. In *WWW*.

[56] Yongfeng Zhang, Xu Chen, Qingyao Ai, Liu Yang, and W Bruce Croft. 2018. Towards conversational search and recommendation: System ask, user respond. In *CIKM*.

[57] Bowen Zheng, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Adapting large language models by integrating collaborative semantics for recommendation. *arXiv preprint arXiv:2311.09049* (2023).

[58] Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V Le, Ed H Chi, et al. 2024. NATURAL PLAN: Benchmarking LLMs on Natural Language Planning. *arXiv preprint arXiv:2406.04520* (2024).

[59] Yaochen Zhu, Liang Wu, Qi Guo, Liangjie Hong, and Jundong Li. 2024. Collaborative large language model for recommender systems. In *WWW*.