WhatNext? - AI-Powered Location Recommendation Application

Eugene Kim eykim@ucsd.edu
Wenzhou Lyu wlyu@ucsd.edu
Chenyang Dong c1dong@ucsd.edu
Jingbo Shang jshang@ucsd.edu

Abstract

In the modern fast-paced social environment, people face significant challenges in planning activities, resulting in wasted time, decision fatigue, and suboptimal choices. This paper introduces our solution: "WhatNext?", a recommendation application that merges real-time data with natural language processing to offer dynamic, personalized suggestions for activities. By leveraging Yelp's location data and OpenAI's Assistant API framework, our app's built-in chatbot ensures that users can quickly find activities and places that are not only open but also align with their specific preferences. It operates through two main functions: parameter inference and contextualized ranking. The recommender system first analyzes the conversation history to generate search parameters for fetching appropriate locations via the API. Then, it systematically ranks these suggestions for the user, drawing from both the dialogue history and a set of personalized preferences. We evaluate our system on two different metrics, proximity/availability and user satisfaction. Initial results reveal overall scores of 0.99 and 0.94, respectively. Our approach demonstrates the potential of AI-integrated systems in improving the efficiency and quality of decision-making.

Code: https://github.com/eugenekim3107/whatnext

1 Introduction

Digital technology has transformed how people plan and engage in social activities. With the increase of data on the internet, people have access to millions of options, including places to eat, events to attend, and things to do. While a large number of options may seem great, it introduces a significant challenge in decision-making. Users often need to navigate through a maze of online information, struggling to make decisions that align with their preferences, location, and time constraints. This not only leads to decision fatigue but also results in sub-optimal choices, reducing the quality of overall user experience. To address these problems, this paper introduces "WhatNext", an iOS app designed to streamline the decision-making process by providing personalized, real-time recommendations for activities.

"WhatNext?" integrates large language models (LLMs) with real-time location data to improve user experiences in activity planning. Utilizing OpenAI's Assistant API for our LLM infrastructure and Yelp's business dataset for location information, the Assistant API serves as a conversational framework that enables function calling. This framework still requires integration with models like GPT-3.5 and GPT-4.0 to operate. Our system's pipeline begins with the user initiating a conversation with ChatGPT by expressing personal preferences. When the user requests recommendations, ChatGPT retrieves relevant locations and prioritizes them based on conversational history and the user's stated preferences. Users can provide feedback to further refine the recommendations. This process forms the backbone of our recommender system.

In our evaluation, experiments with the Proximity and Availability Metric (PAM) and User Satisfaction Metric (USM) produced results of 0.99 and 0.94, respectively. PAM measures the ability of LLMs to meet specific distance and time constraints, while USM measures the effectiveness in delivering recommendations with high user satisfaction. Our application introduces a novel use-case of integrating LLMs with real-time data for personalized recommender systems.

2 Literature Review

The emergence of LLMs has developed a new field of research focused on exploring the potential capabilities of LLMs on a wide range of natural language processing (NLP) tasks. LLMs are trained on massive datasets, which enable them to perform tasks with limited information from user prompts. To evaluate the effectiveness of these models as recommender systems, we first review two related fields: recommender systems and weak supervision.

Recommender Systems: Traditional recommender systems offer suggestions for items based on user behavior, preferences, and interaction data. Initially, these systems utilized explicit feedback, such as user ratings, and simple heuristic algorithms to predict preferences [Resnick and Varian (1997)]. However, with advancements in machine learning, more sophisticated methods have emerged. Techniques like collaborative filtering, content-based filtering, and hybrid models now leverage both user and item data to enhance recom-

mendation accuracy [Sarwar et al. (2001), Pazzani and Billsus (2007), Good et al. (1999)]. Despite their advancements, these systems still struggle with issues such as the cold start problem, where new users or items have limited historical data [Park and Chu (2009)]. LLMs as recommender systems is seen as a possible alternative [Liu et al. (2023)]. LLMs' ability to operate efficiently on minimal information and process explicit user preferences can potentially outperform standard recommender system given new users or items.

Weak Supervision: Weak supervision addresses the challenge of leveraging models to achieve high performance with minimal or noisy input data. This is particularly relevant to our task of using ChatGPT for generating recommendations based on sparse user information. Recent studies have demonstrated methods to enhance GPT-3's performance through strategic prompting techniques. Wang and Lim (2023) developed a three-step prompting approach to enhance GPT-3's recommendation performance. This strategy, which emphasizes recalling user preferences with minimal input, has proven to outperform traditional models that rely on extensive datasets for training. Zhang, Wang and Shang (2023) introduced another form of prompt engineering to indirectly extract information from GPT-3's embeddings. These methods highlight the potential of using advanced prompting techniques with GPT models to achieve better performance, an approach that we aims to refine and implement within our framework.

In summary, the literature highlights LLMs' potential as effective recommender systems, combining conventional recommendation methods with careful prompting strategies. The past work in recommender systems and weak supervision provides a solid foundation for our system to build upon.

3 Methodology

Our app aims to integrate ChatGPT with real-time data and user preferences to optimize the recommender system. We implement our system as a user friendly chat bot, allowing users can provide feedback when necessary. For the frontend development, we use Swift/SwiftUI ensuring a responsive and intuitive user interface for iOS devices. On the backend, we leverage FastAPI, hosted on an AWS EC2 instance, for efficient handling of requests and data processing. We manage user information and location data on MongoDB, session ID on Redis, and profile images on an Amazon S3 bucket. Furthermore, Google Firebase is used for user account management, including login functionalities. This setup outlines our complete system infrastructure.

Our recommender system functions as a loop by using feedback from users to offer better suggestions. It takes both indirect and direct user preference inputs to minimize the cold start problem seen in past papers [Park and Chu (2009)]. Upon account creation, users specify their indirect preferences through the selection of various tags. Additionally, during chat interactions, users can provide direct preferences in their prompts. This data is sent to the server-side for processing by ChatGPT. When a recommendation request is made, ChatGPT fetches location data with inferred parameters to curate a list of suggestions. These suggestions are refined based on the user's conversation history and stated

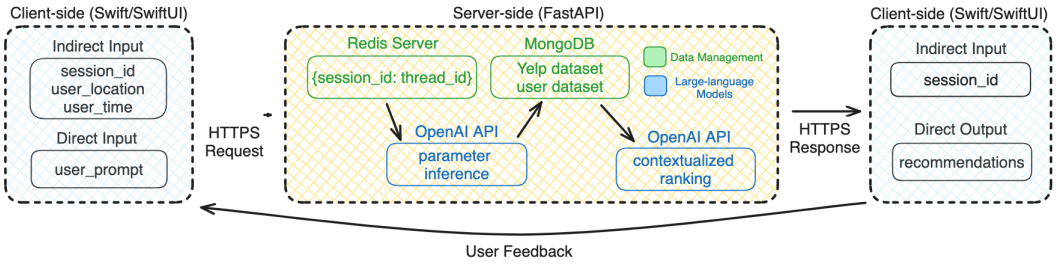

Figure 1: Complete workflow of the recommender system that incorporates user request, system response, and feedback loop.

preferences. Users are encourage to offer feedback for better recommendations. The full workflow of the recommender system is displayed in figure 1.

3.1 Parameter Inference

The OpenAI Assistant API serves as the framework of our system, leveraging its ability to call functions and infer parameters from the chat conversation. It identifies users' preferences for activities, interests, and contextual clues, allowing for the adjustment of search parameters such as business hours, radius, category, and tag (subcategory). This ensures searches are closely aligned with users' current preferences and context.

The API also distinguishes between user queries, recognizing when users are seeking for recommendations, asking for more detail about a certain location, or simply providing personal preferences. This functionality triggers appropriate function calls at necessary moments, improving the app's efficiency by fetching data only when needed. This method highlights the use of conversational AI in enhancing user experiences through contextually appropriate information retrieval.

3.2 Contextualized Ranking

After gathering potential locations through parameter inference, the system progresses to contextualized ranking. This process uses the user's personal tags and information from the conversation to sort recommendations. Locations are selected and prioritized based on how well they match the user's indicated preferences in the prompt and their profile tags. The ranking is adaptive, taking into account the user's proximity, interests alignment, and specific features mentioned in the conversation. After presenting the suggestions to the user, the user has the opportunity to provide feedback for improved recommendations.

By integrating OpenAI's Assistant API for parameter inference and contextualized ranking, our app offers recommendations that incorporate real-time data with personalized preferences.

4 Experiments

4.1 Dataset

Our app requires two main datasets: business and user data. The source and method of data extraction heavily influence the recommender system's performance. We obtain business data from the Yelp API externally, while user data is collected internally.

4.1.1 Business Data

We use Yelp API to access up-to-date information on local businesses, events, and activities. For simplicity, we only collect business data within a 25 mile radius from 9500 Gilman Dr, La Jolla, CA. We store the necessary features for each business in our MongoDB. For a detailed explanation of the selected features, refer to Table 1.

Table 1: Data dictionary for the location features used in our experiments.

Field	Type	Description
business_id	str	Unique identifier for the business.
name	str	The display name of the business.
stars	float	The average rating of the business out of 5.
review_count	float	The number of reviews the business has received.
cur_open	int	Current open status of the business (0 by default).
categories	List[str]	List of categories associated with the business.
tag	List[str]	List of tags associated with the business.
price	str	The price level of the business. Indicated by the number of dollar signs.

4.1.2 User Data

Our system requires two types of user data: the user's current location and their profile tags. We utilize SwiftUI's location services for continuous tracking of the user's current location. For profile tags, the user is required to select a subset of tags upon account creation.

4.2 Evaluation Metrics

To evaluate the effectiveness and user satisfaction of the our recommendation application, we implement two primary metrics: User Satisfaction Metric (USM) and Proximity and Availability Metric (PAM). These two metrics combined serve as a comprehensive measure to assess the application's effectiveness and its alignment with user expectations.

4.2.1 User Satisfaction Metric

The User Satisfaction Metric assesses how well recommendations match user preferences, based on feedback for system-generated recommendations. User evaluations are classified into three categories: positive, neutral, or negative, with assigned weights of 1, 0.5, and 0, respectively. The User Feedback Score (UFS) is then calculated by taking the weighted sum of all feedback and dividing by the total feedback count:

$$UFS = \frac{\sum_{i=1}^{N} W_i}{N} \tag{1}$$

In this formula, W_i corresponds to the weight assigned to the i^{th} feedback category, and N is the total count of feedback received. This metric offers a direct quantitative measure of user satisfaction.

4.2.2 Proximity and Availability Metric

Aside from the user satisfaction-based metric, we also use PAM to offer a more balanced evaluation of the application's performance. The primary objective is to assess the practicality of the recommendations provided by the application. This metric takes into account two main criteria: the business hours of the recommendations and the distance between the user and recommendations. If the recommended location is within the desired proximity and satisfies the time constraints specified by the user, it will receive a PAM score of 1, else it will receive 0. The Recommendation Relevance Score (RRS) is calculated by the sum of PAM for each location divided by the total number of locations:

$$RRS = \frac{|N_P \cap N_A|}{N} \tag{2}$$

where N_P and N_P represents the number of recommendations that satisfy the proximity and availability criteria respectively. N denotes the total number of recommendations.

4.3 Implementation

In our experiment, we evaluate the effectiveness of generating recommendations using structured and semi-structured prompts across two LLMs: gpt-3.5-turbo-0125 and gpt-4-0125-preview. This approach enables comparison of model performances and calculation of the rule-based PAM and preference-based USM. Our experiment utilizes four types of parameters: categories, tags (sub-categories), business status, and distance. We select seven categories: hiking, fitness, beautysvc, food, restaurant, shopping, and aquariums. Three tags are applied with each category, with business status set to "currently open" and distance parameters defined as less than 10 km or less than 20 km from the user. Tag distribution is presented in Table 2. Tags for hiking and aquariums are not used in this experiment due to the lack of data.

Table 2: Distribution of Tags for Selected Categories

Category	Tags
Fitness	yoga, gyms, dancestudio
Beautysvc	skincare, waxing, hair
Food	coffee, breakfast_brunch, sandwiches
Restaurant	breakfast_brunch, sandwiches, seafood
Shopping	jewelry, florists, kitchenandbath

We use both structured and semi-structured prompts for testing. Structured prompts are precisely formulated queries that specify all experiment parameters required for the model to generate a response. This is used to increase control over the experiment and decrease the effect of confounding variables. They explicitly state what information is needed, including categories, status, tags, and distances in recommendation systems. This type of prompt requires the model to adhere strictly to the provided parameters, yielding very specific and targeted responses.

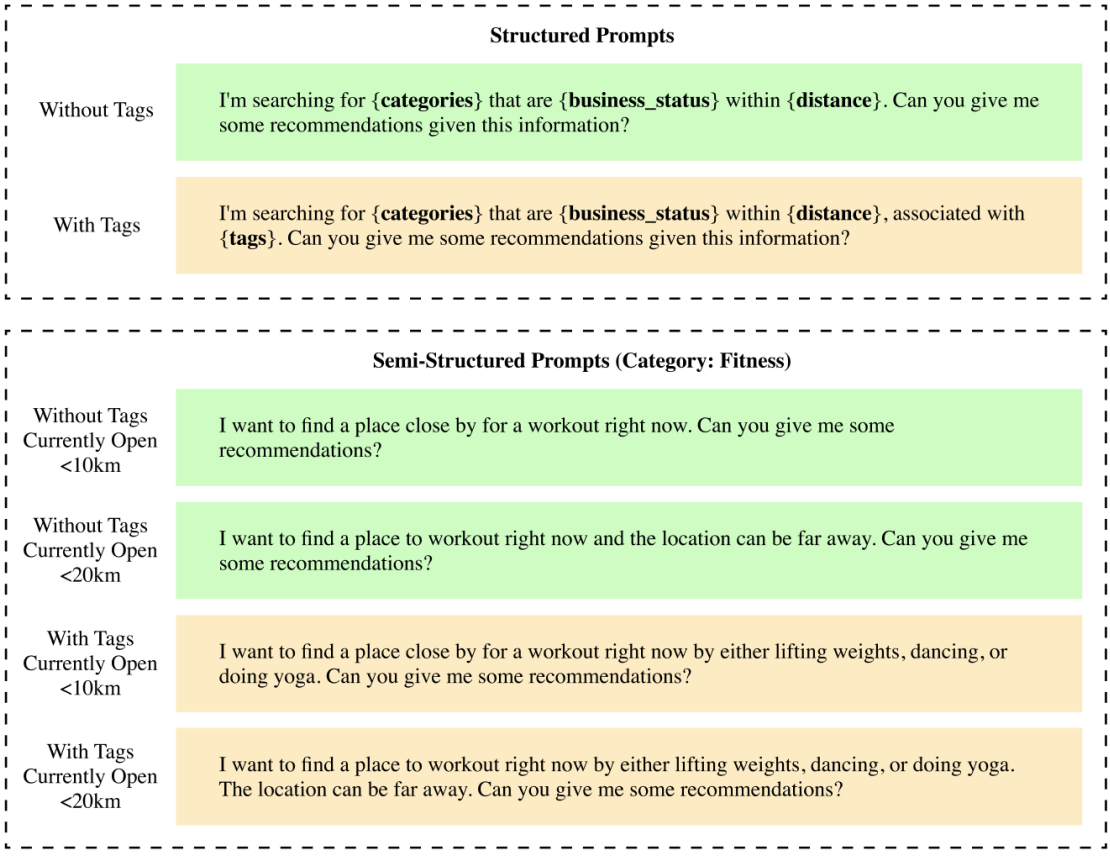

Figure 2: Example of structured and semi-structured prompts on fitness category

The semi-structured prompts are employed to test the models' parameter inference capabilities. These prompts provide certain parameters while allowing the model to infer additional details, aiming to imitate realistic human queries. This method tests the models' ability to

understand and fill in contextual gaps based on partial information, reflecting real-world use cases where users may not always provide complete details. For an example of the prompts, refer to Figure 2.

5 Results

We present the findings from a comparative analysis of GPT-3.5 and GPT-4. Table 3 records the Proximity and Availability metrics, where GPT-4 consistently achieves a score of 1.0 across all categories for structured and semi-structured prompts. GPT-3.5 shows good performance with a mean score of 0.98 for structured prompts and 1.0 for semi-structured prompts.

Table 4 shows an improvement from GPT-3.5 to GPT-4, with GPT-4 achieving higher average scores—0.97 for structured prompts and 0.96 for semi-structured prompts—compared to GPT-3.5's scores of 0.91 and 0.9, respectively. This demonstrates GPT-4's enhanced ability to infer parameters, making it better suited for practical applications where users often omit information.

Moreover, the higher overall score observed in Table 3 (0.99) compared to Table 4 (0.94) is due to the use of rule-based algorithms. These algorithms allow for precise control and filtering of parameters like business operation hours and distance, promoting consistency and minimizing variability. In contrast, the scores in the latter table measures user satisfaction, which is subject to individual expectations and experiences.

Table 3: Proximity and Availability Metric Results

Category	Structured Prompts		Semi-structured Prompts		
	GPT-3.5	GPT-4.0	GPT-3.5	GPT-4.0	All
Hiking	1.0	1.0	1.0	1.0	1.0
Fitness	0.93	1.0	1.0	1.0	0.98
Beauty & Spas	1.0	1.0	1.0	1.0	1.0
Food	0.9	1.0	1.0	1.0	0.98
Restaurant	1.0	1.0	1.0	1.0	1.0
Shopping	1.0	1.0	1.0	1.0	1.0
Aquariums	1.0	1.0	1.0	1.0	1.0
All	0.98	1.0	1.0	1.0	0.99

Table 4: User Satisfaction Metric Results

Category	Structured Prompts		Semi-structured Prompts		
	GPT-3.5	GPT-4.0	GPT-3.5	GPT-4.0	All
Hiking	1.0	1.0	1.0	1.0	1.0
Fitness	0.89	1.0	0.85	0.99	0.93
Beauty & Spas	0.95	0.99	0.90	0.99	0.96
Food	0.92	0.98	0.90	1.0	0.95
Restaurant	0.88	0.99	0.90	1.0	0.94
Shopping	0.84	0.93	0.98	0.86	0.90
Aquariums	0.92	0.88	0.80	0.88	0.87
All	0.91	0.97	0.9	0.96	0.94

6 Conclusion

Based on the results of our experiments, ChatGPT serves as an effective recommendation system, surpassing conventional search engines by handling typos, incorporating user bios, and integrating feedback. Its ability to call functions and infer parameter settings based on the conversation simplifies user interaction. In conclusion, our system enhances the decision-making experience by minimizing decision fatigue and delivering tailored recommendations.

References

Good, Nathaniel, J Ben Schafer, Joseph A Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, John Riedl et al. 1999. "Combining collaborative filtering with personal agents for better recommendations." *Aaai/iaai* 439 (10.5555): 315149–315352

Liu, Junling, Chao Liu, Renjie Lv, Kang Zhou, and Yan Zhang. 2023. "Is chatgpt a good recommender? a preliminary study." *arXiv preprint arXiv:2304.10149*

Park, Seung-Taek, and Wei Chu. 2009. "Pairwise preference regression for cold-start recommendation." In *Proceedings of the third ACM conference on Recommender systems*.

Pazzani, Michael J, and Daniel Billsus. 2007. "Content-based recommendation systems." In *The adaptive web: methods and strategies of web personalization*. Springer: 325–341

Resnick, Paul, and Hal R Varian. 1997. "Recommender systems." *Communications of the ACM* 40 (3): 56–58

Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl. 2001. "Item-based collaborative filtering recommendation algorithms." In *Proceedings of the 10th international conference on World Wide Web*.

Wang, Lei, and Ee-Peng Lim. 2023. "Zero-Shot Next-Item Recommendation using Large Pretrained Language Models." *arXiv preprint arXiv:2304.03153*

Zhang, Yuwei, Zihan Wang, and Jingbo Shang. 2023. "ClusterLLM: Large Language Models as a Guide for Text Clustering." *arXiv preprint arXiv:2305.14871*

Appendices

A.1 Cost Analysis

The cost of maintenance for ”WhatNext?” per month is $6944.86 for gpt-4-0125-preview and $361.36 for gpt-3.5-turbo-0125. This cost assumes a total of 1000 active users who make 20 API calls per day on average. The complete cost calculation is display in Table A 1.

Table A 1: Monthly Cost Analysis for 1000 Users with 20 API Calls/Day/User

Service	gpt-3.5-turbo-0125	gpt-4-0125-preview
EC2	$11.17	$11.17
EBS	$3.00	$3.00
S3	$0.694	$0.694
API Calls	$346.50	$6,930.00
Total	$361.36	$6,944.86