Restaurant Recommender AI

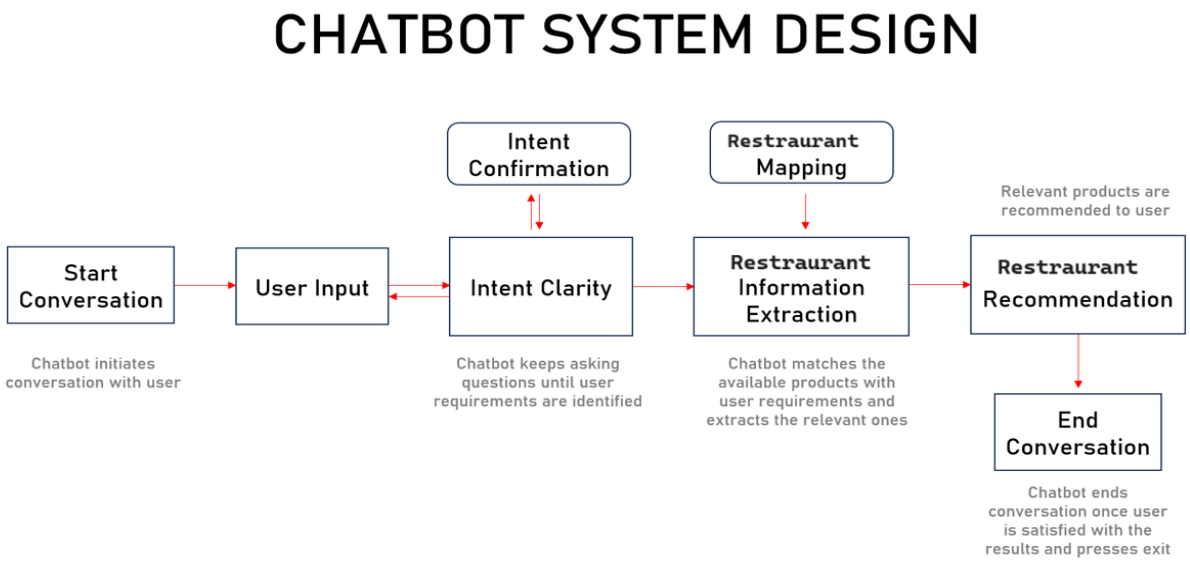
Project Overview

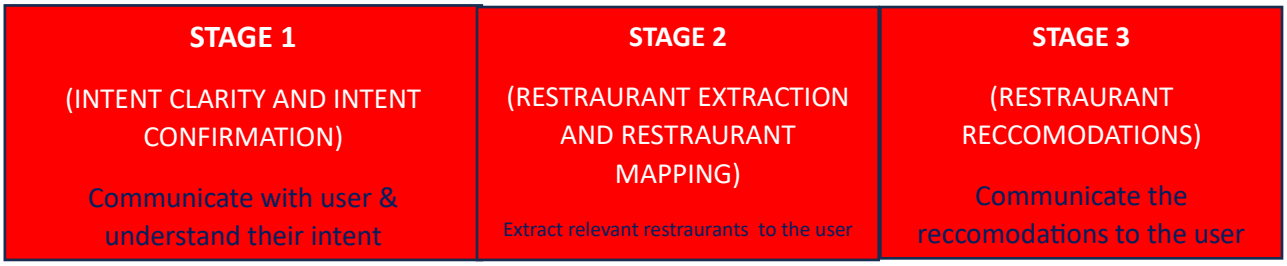
Finding the perfect restaurant can be challenging due to the vast number of choices available. To help users make informed decisions, we have developed Restaurant Recommender AI, a chatbot designed to assist users in finding the ideal restaurant based on their specific needs, such as cuisine preferences, dietary restrictions, location, and user reviews. This solution leverages both large language models (LLMs) and rule-based functions to provide personalized recommendations.

Objective

1. Develop a chatbot that can accurately parse a dataset containing comprehensive information about restaurants.
2. Provide personalized restaurant recommendations based on user requirements.
3. Understand and process user preferences such as cuisine type, dietary restrictions, location, and user reviews.
4. Suggest the best dining options tailored to individual user needs.
5. Understanding the multi-stage architecture behind the fully functioning chatbot system,
6. Integrate OpenAI's Chat Completion APIs to provide a natural and contextually relevant conversation.

System Design





As you can see in this image, there are three stages of the chatbot, which are as follows:

- **Stage 1:** Intent Clarity and Intent Confirmation
- **Stage 2:** Product Extraction and Product Mapping
- **Stage 3**: Product Recommendation.

Stage 1 - Intent Clarity and Intent Confirmation

The first stage involves a conversation between the user and the AI system. Python functions such as **initialize_conversation()**, trigger the conversation, and those such as **get_chat_completions()** allows the conversation to continue with each conversation via LLM calls.
The stage includes an additional layer called **moderation_check()** to flag and discontinue conversations that contain unsafe or sensitive content.

The output of this stage is a dictionary that has captured all the needs of the user (User Requirements Dictionary) in key-value pairs or specifically JSON. The reason for JSON output is for output parsing convenience - the AI system aims to collect user requirements and store them in a Python dictionary for further processing rather than a string. It should be noted that LLMs are token producers and output tokens are primarily in string format rather than Python objects, unless specifically mentioned.
Therefore, to guard the scenario where we do not have a dictionary output, a Python function called 'dictionary_present()' is created. It is used to convert the LLM output, which might resemble a dictionary in string format, into an actual Python dictionary.

The overall functioning of the chatbot in this stage happens in the following steps:

Initialising conversation and chat model completion: The conversation starts with the **'initialize_conversation()'** function, where the AI system introduces itself and asks for the user's requirements. The subsequent messages from the user and the AI system are attributed to **'chat_model_completion()**' function where the OpenAI's chat completion API is utilised for continuing the user conversation, until the requirements have been completely identified.

Intent confirmation: The AI system uses an **'Intent Confirmation layer' (intent_confirmation_layer())** as a flag (yes or no) to indicate whether all requirements of the user have been captured. During the conversation, if the system

receives ‘no,’ it understands that further questions are needed to capture all the requirements.

User requirements dictionary: Once the **‘Intent Confirmation’ layer** confirms that all requirements of the user have been captured (*i.e.* ‘yes’ flag), it passes the requirements to the ‘User Requirements Dictionary’ (**dictionary_present() function**) to be converted into a dictionary/ dictionary-like object to the **user_req** variable.

Stage 2 - Product Mapping and Information Extraction

The second stage of the system is the ‘Product Mapping and Information Extraction’ stage. This stage filters the restaurant s as per the user requirements dictionary captured in the previous stage and uses it to present the top three restaurant recommendations to the user. The illustrations below shows functions and steps in Stage 2.

The entire restaurant filtering process in Stage 2 can divided into two parts:
Part 1 (Product Mapping): You create a feature dictionary for each restaurant from its given description. This is done using the **product_map_layer()** function, which extracts key features and criteria from restaurant descriptions.

This function extracts the primary restaurant features from the detailed description of any restaurant (such as City, Price range Price range, Has Table booking,) speed, etc.). These entries are stored as key-value pairs of a dictionary **res_spec**.

Once these values are extracted, they are mapped with the appropriate classification value **(low, medium or high)** defined inside rules.

To perform this mapping, you need a detailed set of rules from your heuristics, which the LLM model is not privy to.

Since this operation is independent from any user input, you need to execute this function once for all the restaurant s.

Part 2 (Information Extraction): So, till this stage, you have two dictionaries: **res_spec** and **user_req**. All you need to do now is to determine how similar a restraurant ’s features (stored in res_spec) are to the user’s requirements (stored in **user_req**).
But before you compare these two dictionaries, they need to be converted from a string of dictionary to just a dictionary. This is done using the **dictionary_present()** function.Then, they are passed to a rule-based function **compare_restraurant s_with_user()**.

For each feature, a score of 1 is assigned if the feature is the same or better than the user's requirement.Otherwise, a score of 0 is assigned.
The scoring is performed for all the restaurant s iteratively. Once the scoring is done, the scores are then used to rank and identify the top three restaurant s as recommendations for the user.

This scoring is not fixed, as it can be modified as per your requirement for any particular use case.

Product Validation Layer: Once the top three restaurant s are extracted, the list is sent to the product validation layer, which ensures that only relevant products are forwarded as recommendations.
The purpose of the product validation layer is to ensure that only restaurant s with a **score of three or above** are recommended to the user. The score threshold of three is arbitrary but is chosen based on the desire for at least three features to meet or exceed the user's requirements.
So, if a restaurant 's total score is greater than two (indicating that it meets or exceeds user's requirements in at least two features), it is considered a recommendation.
The images below shows the entire product mapping and product information extraction for two sets of **res_spec** variables.

Stage 3 - Product Recommendation

Finally, you have reached the product recommendation layer. It takes the output from the 'compare_restraurants_with_user' function in the previous layer and provides the recommendations to the user. The broader process happening in this stage can be summarised as follows:

The product validation layer will recommend a maximum of three restaurant s to the product recommendation layer. This ensures that the user is presented with a manageable number of restaurant options to choose from.

It is possible that no restaurant s meet the score threshold of three, in which case the product validation layer will feed 'None' or 'No restaurant s matched' to the product recommendation layer. In such cases, the AI system will be instructed to connect the user to a human expert.

Implementation Challenges

Building a restaurant recommender AI using OpenAI APIs involves several key challenges:

1. Data Collection and Quality
    - Diverse Data Sources: Aggregating accurate, up-to-date data from sources like Yelp and Google Maps.
    - User Preferences: Accurately interpreting and incorporating user preferences.

2. Natural Language Understanding
   - Contextual Understanding: Processing natural language queries accurately.
   - Ambiguity Resolution: Handling ambiguous and diverse language inputs.

3. Personalization and Relevance
   - User Profile Building: Developing robust user profiles.
   - Dynamic Adaptation: Continuously adapting to user preferences.
   - Balancing Preferences: Managing conflicting user preferences.

4. Scalability and Performance
   - Real-time Processing: Handling real-time queries efficiently.
   - Handling Large Datasets: Managing large volumes of data.
   - API Rate Limits: Operating within API rate limits.

5. Integration with External Systems
   - API Compatibility: Seamless integration with external APIs.
   - Data Consistency: Maintaining consistency and integrity.
   - Security and Privacy: Ensuring secure and compliant data handling.

6. Recommendation Algorithms
   - Algorithm Selection: Choosing appropriate recommendation algorithms.
   - Bias Mitigation: Ensuring algorithms are free from biases.
   - Contextual Recommendations: Considering context such as time and location.

7. Evaluation and Feedback Loop
   - Performance Metrics: Defining and tracking key metrics.
   - User Feedback Integration: Continuously improving based on feedback.
   - A/B Testing: Comparing different strategies for effectiveness.

8. User Experience and Interface
   - Intuitive Interface: Designing a user-friendly interface.
   - Multi-Platform Support: Ensuring compatibility across platforms.
   - Clear Explanations**: Providing transparent recommendation explanations.

9. Legal and Ethical Considerations
   - Compliance: Adhering to legal requirements and privacy regulations.
   - Ethical Recommendations: Avoiding harmful or biased recommendations.
   - Transparency: Being transparent about data usage.

10. Flow Design and API Integration
   - Model Flow Design: Creating an efficient flow from data input to recommendation output.
   - API Integration: Effectively integrating multiple APIs for seamless data retrieval and processing.

Successfully addressing these challenges requires a multidisciplinary approach involving machine learning, data engineering, natural language processing, and UX design.

Lessons Learned

Throughout the development of the restaurant recommender AI, several key lessons emerged. Firstly, the importance of high-quality, accurate, and up-to-date data became evident, as it is crucial for reliable recommendations. Accurately collecting and interpreting user preferences significantly improved personalization. Advanced NLP models proved essential for processing and understanding user queries, while robust handling of ambiguous and diverse language inputs enhanced user satisfaction. Continuously updating user profiles led to better-tailored recommendations, and effectively managing conflicting preferences was key to maintaining user trust. Efficient real-time processing was necessary to handle large volumes of data and queries, and designing within API rate limits ensured system functionality. Effective integration with external APIs provided comprehensive data access, and maintaining consistency and integrity across systems was essential. Implementing strategies to mitigate biases in recommendation algorithms and factoring in contextual elements like time and location improved relevance. Defining and tracking key performance metrics helped assess system effectiveness, and incorporating user feedback along with conducting A/B testing drove ongoing enhancements. A user-friendly interface significantly enhanced engagement, while ensuring multi-platform support broadened accessibility. Adhering to legal and privacy regulations was mandatory, and avoiding biased or harmful recommendations maintained ethical standards. Being transparent about data usage built user trust.

Outcomes Achieved

The development process yielded several positive outcomes. Enhanced user satisfaction was achieved through improved accuracy and relevance of recommendations, driven by high-quality data and advanced NLP. Dynamic personalization and context-aware suggestions further improved user engagement. Operational efficiency was realized through efficient real-time processing capabilities, enabling smooth handling of large query volumes. Robust integration with multiple external APIs provided comprehensive data access. System robustness was bolstered by bias mitigation strategies, resulting in fairer recommendations, and effective management of API rate limits ensured uninterrupted service. Continuous improvement was facilitated by regular updates based on performance metrics and user feedback, with successful A/B testing refining recommendation strategies. Compliance with legal and privacy standards ensured user data protection, while transparent recommendation processes fostered user trust and confidence. Overall, the development process underscored the importance of data quality, efficient integration, advanced NLP, and a user-centric approach, leading to a robust and effective restaurant recommender AI.