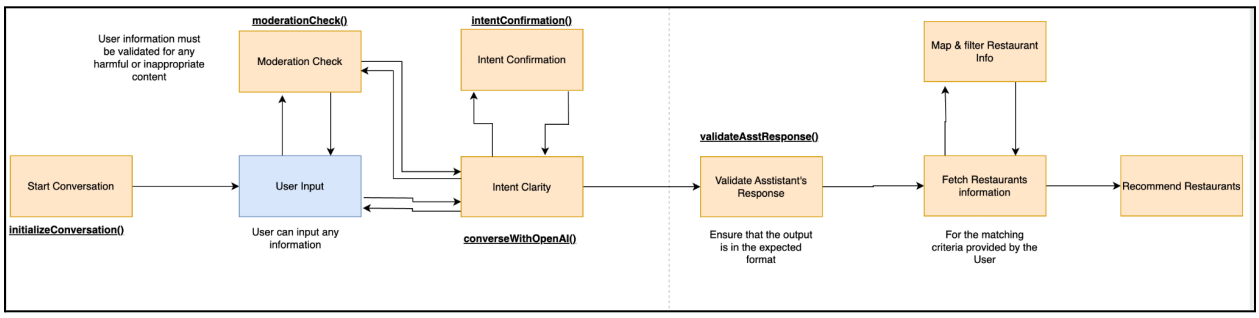
Project Report: Restaurant Recommendation Chatbot Powered by AI

1. Objective

The primary objective of this project is to design and implement an **AI-powered chatbot** that recommends restaurants based on user preferences. The chatbot leverages natural language queries (e.g., *"Find me vegetarian restaurants under ₹500 in Indiranagar"*) and provides tailored recommendations using structured restaurant datasets combined with OpenAI's language model capabilities.

---

2. Design



System Architecture

The chatbot system is structured as follows:

1. **User Input Layer**
    - Users interact with the chatbot through natural language queries.
2. **Preprocessing & Data Layer**
    - A dataset (`restaurant_dataset.csv`) containing restaurant information (cuisine, price, rating, location, etc.) is loaded and processed with **Pandas**.

3. **AI Model Integration**
    - The **OpenAI Chat Completion API** is used to handle conversation, extract intent, and generate human-like responses.
    - The system message defines the chatbot's role: *"You are a restaurant recommendation assistant."*
4. **Recommendation Engine**
    - Filters restaurants based on extracted constraints such as cuisine, budget, and location.
    - Formats recommendations to present to the user.
5. **Output Layer**
    - Chatbot returns clear, conversational responses containing a list of restaurants matching the request.

---

3. Implementation

3.1 Importing Required Libraries

- `pandas`: Used to load and manipulate the restaurant dataset.
- `openai`: To connect with the OpenAI Chat Completion API and generate chatbot responses.
- `json`: For structured data representation when passing responses.

This ensures the environment is set up for both **data handling** and **AI-powered conversation**.

3.2 Loading the Dataset

- A CSV file containing restaurant information was loaded using `pandas.read_csv()`.

- Columns included attributes such as:
    - `Name` (restaurant name)
    - `Cuisine` (e.g., Italian, Chinese, Indian)
    - `Location` (area/city)
    - `Price` (average cost for two people)
    - `Rating` (user ratings out of 5)

  This provides a structured **knowledge base** for the chatbot.

3.3 Exploring and Cleaning Data

- Inspected the dataset (`df.head()`, `df.info()`, `df.describe()`) to understand structure.
- Checked for **null values or inconsistencies**.
- Converted certain columns to consistent formats (e.g., converting all cuisines to lowercase, ensuring numeric fields like price are integers).

  This makes filtering reliable later on.

3.4 Filtering Logic (Recommendation Engine)

- Functions were written to filter restaurants based on user constraints such as:
    - **Cuisine** (e.g., “Chinese”)
    - **Location** (e.g., “Indiranagar”)
    - **Price range** (e.g., under ₹500)
    - **Ratings** (e.g., >4.0)

  This acts as the **retrieval engine**.

3.5 Integrating with OpenAI API

- OpenAI's **Chat Completions API** (`gpt-3.5-turbo`) was integrated.
- Chatbot roles were defined:
    - **System role**: Sets context → "You are a restaurant recommendation assistant."
    - **User role**: User's input query.
    - **Assistant role**: LLM generates conversational response.

      This ensures the **chatbot understands natural language**.

3.6 Combining Dataset Lookup + AI Responses

- When the user asks, the chatbot:
    1. Extracts intent (cuisine, budget, location).
    2. Queries the dataset using the filtering function.
    3. Passes the filtered results to OpenAI to generate a **well-phrased conversational response**.

4. Challenges

1. **Dataset Limitations**
    - The dataset may lack certain fields (e.g., ambience, parking, dietary preferences). This limits the richness of recommendations.
2. **Handling Ambiguity in User Input**
    - Natural language inputs like *"a cheap cozy place for date night"* are hard to map directly to structured dataset filters.
3. **API Dependency**
    - The chatbot depends heavily on OpenAI API availability and cost, which could be limiting in production-scale scenarios.

4. **Scalability**
    - Current implementation works for small datasets but may not handle very large datasets efficiently without optimization (e.g., indexing, embeddings).

---

5. Lessons Learned

1. **Combining AI with Structured Data**
    - Pure LLM-based chatbots are good at conversation, but grounding them in **real datasets** makes recommendations accurate and reliable.
2. **Importance of Context Management**
    - Defining system prompts carefully (e.g., “You are a restaurant recommendation assistant”) helps keep the chatbot focused on its domain.
3. **Need for Moderation & Intent Filtering**
    - Moderation ensures safe interaction, while intent filtering makes sure the chatbot stays relevant to restaurant recommendations.
4. **Future Enhancements**
    - Adding personalization (user history).
    - Using embeddings for semantic search (e.g., “romantic rooftop places”).
    - Expanding dataset fields for richer filtering (dietary restrictions, ambience, services).