Restaurant Recommender AI Chatbot - Project Report

---

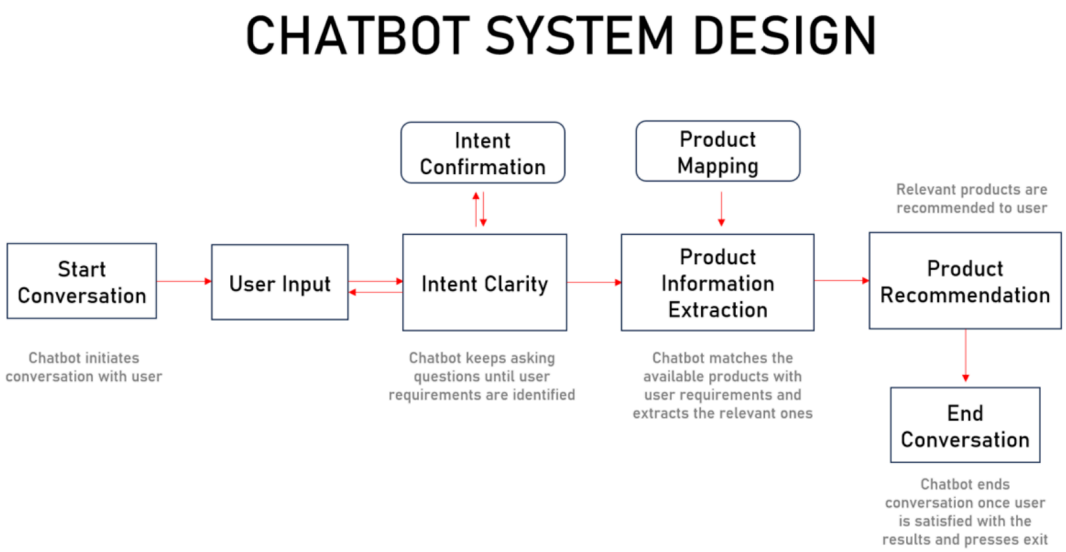
1. Project Overview

Objective: To develop a conversational AI chatbot that recommends restaurants based on user preferences such as country, city, preferred cuisine, and budget, leveraging a clean dataset from Zomato and OpenAI's language model.

Outcome: A fully functional chatbot that collects user requirements through natural dialogue, filters appropriate restaurants from a dataset, and presents top recommendations.

2. System Architecture

Architecture Diagram:



As shown in the image, the chatbot contains the following layers:

- Intent Clarity Layer
- Intent Confirmation Layer
- Product Mapping Layer
- Product Information Extraction Layer
- Product Recommendation Layer

Major functions behind the Chatbot:

Let's now look at a brief overview of the major functions that form the chatbot. We'll take a deep dive later

- `initialize_conversation()`: This initializes the variable conversation with the system message.
- `get_chat_model_completions()`: This takes the ongoing conversation as the input and returns the response by the assistant
- `moderation_check()`: This checks if the user's or the assistant's message is inappropriate. If any of these is inappropriate, it ends the conversation.
- `intent_confirmation_layer()`: This function takes the assistant's response and evaluates if the chatbot has captured the user's profile clearly. Specifically, this checks if the following properties for the user has been captured or not
- Country, City, Cuisine, Price Preference
- `dictionary_present()`: This function checks if the final understanding of a user's profile is returned by the chatbot as a python dictionary or not. If there is a dictionary, it extracts the information as a Python dictionary.
- `compare_restaurant_with_user()`: This function compares the user's requirements with the different restaurants and their details present in the dataset and comes back with the top 5 recommendations.
- `initialize_conv_reco()`: Initializes the recommendations conversation

Also built a UI for chatbot using Flask. Below are the modules and components of Flask App

Modules:

- **Flask Web App:** Backend server to manage chat interactions.
- **OpenAI API Integration:** Collects and understands user requirements conversationally.
- **Dataset Handling:** Zomato restaurant dataset cleaned, merged with country codes.
- **Ngrok Tunnel:** Exposes the local server to the internet for testing.
- **User Interface:** Simple conversational flow using HTML templates.

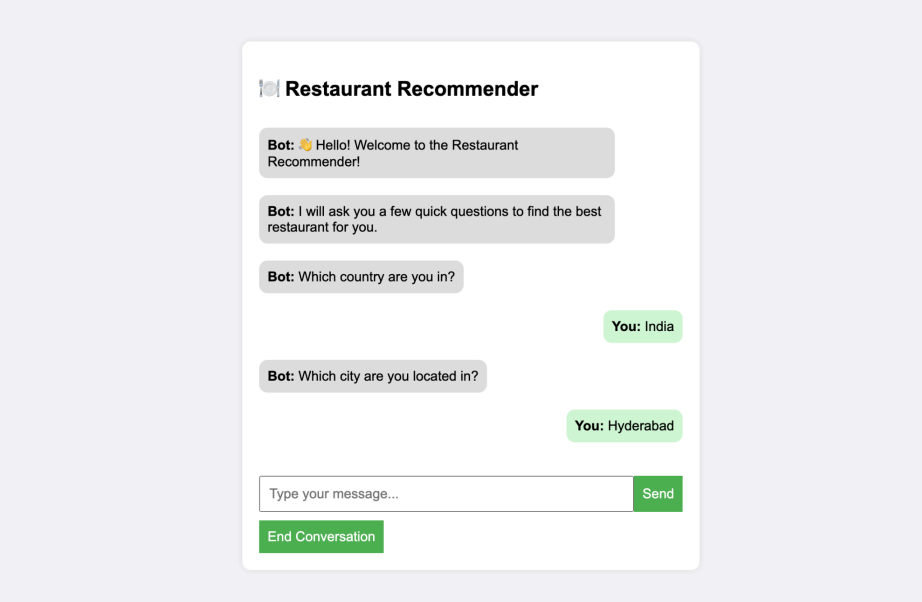
Components:

- `functions.py` / `functions_flask.py`: Utility functions for conversation, moderation, matching.

- `app.py`: Main Flask app managing routes and session states.

- `index.html`: Frontend chat interface.

How to run the Flask app?

1. Download project folder from github (Link: https://github.com/supriyaKU23/restaurant_recommender_project)
2. Make sure pip3 and python3 are installed in your system
3. Open terminal
4. -cd restaurant_recommender_project
5. Run - pip3 requirements.txt
6. Run - python3 app.py
7. You will get output as below
8.
```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.1.13:5000
Press CTRL+C to quit
```
9. Now open http://127.0.0.1:5000/ in the browser. You will see chatbot UI as below
10. 

3. Design Details

Dataset Preparation:

- Loaded Zomato restaurant dataset and Country Codes.
- Merged based on `Country Code`.
- Removed missing names and cuisines.
- Selected necessary fields for recommendations.

Chatbot Flow:

1. Greet user.
2. Ask questions sequentially: Country → City → Cuisine → Budget.
3. Confirm captured details.
4. Match against restaurant database.
5. Display top 5 recommendations sorted by price match and rating.
6. Offer users to restart or exit.

Key Technologies Used:

- **Python (Flask)**: Server-side application.
- **OpenAI GPT-3.5-Turbo**: Natural conversation and understanding.
- **Pandas**: Dataset manipulation.
- **HTML/Jinja2**: User interface templates.

4. Implementation

Main Files:

- `functions_flask.py`: Defines
    - `initialize_conversation()`
    - `get_chat_model_completions()`
    - `moderation_check()`
    - `intent_confirmation_layer()`
    - `dictionary_present()`, `extract_dictionary_from_string()`
    - `product_map_layer()`, `compare_restaurant_with_user()`
    - `initialize_conv_reco()`
    - `check_exit()`
- `app.py`:
    - Flask routes and session state management.
    - Handles different conversation states: `welcome`, `asking_questions`, `ask_more`.
    - After collecting inputs, calls dataset matching logic.

5. Challenges Faced

Challenge	Solution
Colab environment complicates running Flask server	Ran in localhost
Notebook .ipynb corrupted .py exports	Rewrote pure `.py` files manually
Handling incomplete user inputs	Forced question sequence to ensure complete information

OpenAI API upgrades (ChatCompletion not subscriptable)	Updated OpenAI library usage correctly

6. Lessons Learned

- Always separate Jupyter notebook (.ipynb) and pure Python (.py) projects.
- Flask requires a clean app structure without hidden JSON formatting.
- Properly managing session states ensures smoother multi-turn conversation.
- Dataset preprocessing (cleaning, merging) significantly affects recommendation quality.
- OpenAI's conversation models require structured prompts and careful flow planning.

7. Other Challenges Faced

Chatbot can cater to irrelevant requests from the users apart from restaurant assistance

- It can cater to offensive/prohibited content generated from both user and assistant through moderation checks
- Data has a limitation that city or country might not be present in dataset
- One limitation of the chatbot is that it looks for the exact value entered by the user in the database and returns if there are matching rows in the data. It is unable to recommend any alternatives to keep the user engaged.
- At the end, after all the details are fetched, user needs to enter exit to end the conversation
- Chatbot can handle a scenario where the user provides the requirements that do not match with the dataset and replies appropriately.
- One particular challenge faced was when I tried to increase the dataset size, the OpenAI timeout error was quite frequent.
- Sometimes the bot still times out when trying to fetch the results, therefore please try again should you face any such errors.

8. Future Improvements

- Enhance UI with richer HTML/CSS or JavaScript frontend.
- Add user authentication for saving past conversations.
- Integrate Google Maps API for showing restaurant locations.
- Add filtering by ratings, delivery options, and vegetarian-friendly tags.
- Extend chatbot to book tables or show live menus.

8. Conclusion

This project successfully developed a fully functional conversational restaurant recommender using Flask, OpenAI, and Pandas. The bot efficiently understands user requirements, processes a cleaned dataset, and provides top restaurant recommendations through a simple and natural interaction flow.

9. References

- OpenAI API Documentation
- Flask Documentation
- Pandas Library
- Zomato Dataset (Kaggle)

⭐ **End of Report**