RAG Based ChatBot Report

1)Scraped Dataset

a]Overview

The dataset contains structured information scraped from Zomato’s Kanpur dine-out listings. It includes restaurant metadata, contact details, facilities, top dishes, and individual dish information.

b]Data Schema

Each restaurant object contains:

```
{
  "Name": "string",
  "Address": "string",
  "Opening Hours": "string | null",
  "Contact Number": "string",
  "Delivery Rating": "string",
  "Cuisines": "string",
  "Remarks": "string",
  "Facilities": "string",
  "Top_dishes": "string",
  "Dishes": [
    {
      "Dish": "string",
      "Description": "string",
      "Price": "string"
    },
    ...
  ]
}
```

c]Data Collection Methodology

- **Tool used**: Selenium WebDriver
- **Target URL**: `https://www.zomato.com/kanpur/dine-out`
- **Process**:
    - Top 10 restaurant links collected from the listing page.

    - Navigated to each restaurant’s info and order page.
    - Extracted details using class and CSS selectors.
    - Saved data as JSON using Python’s `json` module.

2)Technical Documentation

a]System Architecture

User → Streamlit UI → RetrievalQA Chain (LangChain)
↓
Hugging Face Model FAISS Vector Store

b]Components

- **Frontend**: Streamlit
- **LLM**: `mistralai/Mistral-7B-Instruct-v0.3` (Hugging Face)
- **Embeddings**: `sentence-transformers/all-MiniLM-L6-v2`
- **Vector Store**: FAISS (local)
- **Pipeline**:
    - `scrape_data.py`: Scrapes data
    - `kb.py`: Formats and embeds data into FAISS
    - `bot.py`: Streamlit chatbot using LangChain’s RetrievalQA

c] Implementation Details

- User queries embedded and matched via FAISS similarity search
- Data retrieved and passed into the Mistral model
- Responses generated via LangChain’s `RetrievalQA(chain_type='stuff')`

d]Design Decisions

- **FAISS** used for fast semantic search
- **LangChain** used to simplify chaining retriever + LLM
- **MiniLM** used for compact, fast embeddings
- **Mistral 7B** chosen for instruction-following ability

3)Challenges and Solutions

Challenge	Solution
Zomato uses dynamic JavaScript rendering	Used Selenium with delays and careful selector handling
Missing fields on some restaurant pages	Wrapped extraction in `try/except` for robustness
FAISS deserialization error	Enabled `allow_dangerous_deserialization=True` with local safety checks
Large model load times	Cached model and vectorstore in Streamlit

4)Future Improvements

- We can **add more cities** and retrain embeddings dynamically.
- History can be saved for particular user, and his wishes or suggestions can be used as feedback
- Add **voice input** using speech-to-text for better accessibility.

- And replace the current model with a **smaller, faster quantized version** for mobile deployment.