COMP4332 / RMBI 4310 (Spring 2025) Project 3: Simple LLM Agent-based Conversational Recommender System

TA:

Chunyang LI (cliei@connect.ust.hk)

Qing ZONG (qzong@connect.ust.hk)

Background

LLM Agent

A Large Language Model (LLM) agent is a system that leverages the natural language understanding and generation capabilities of LLMs to perform specific tasks. In this project, you will implement LLM-based agents to process user queries, recognize intentions, parse metadata and summarize reviews for a conversational recommender system.

Conversational Recommender System (CRS)

A Conversational Recommender System (CRS) interacts with users through natural language to provide personalized recommendations. This project focuses on a restaurant recommendation system that processes user queries to provide ratings, comparisons, or review summaries based on restaurant data.

Task Description

Objective

You will implement a simple CRS using LLM agents to handle three types of user queries:

1. **Rating Queries**: Retrieve and calculate the overall rating of a specified restaurant (e.g., "What is the rating of Beyond Falvours?").
2. **Comparison Queries**: Compare two restaurants based on their overall ratings (e.g., "Beyond Flavours and Karachi Cafe, which one is better?").
3. **Review Query**: Summarize the reviews of a specified restaurant (e.g., "What are people saying about Beyond Flavours?").

The system uses a provided dataset (`reviews.csv`) and a code framework (`main.py`) to process queries and generate responses. All the code you need to implement is in `main.py` and labeled with `TODO` comments.

After implementing the required functions and agents, you can try this system with your own queries by running the command:

```
python main.py
```

You can interact with the system in the console (your query should contain the correct restaurant name in the dataset). The system will respond with the generated response. Here is an example of how the system works:

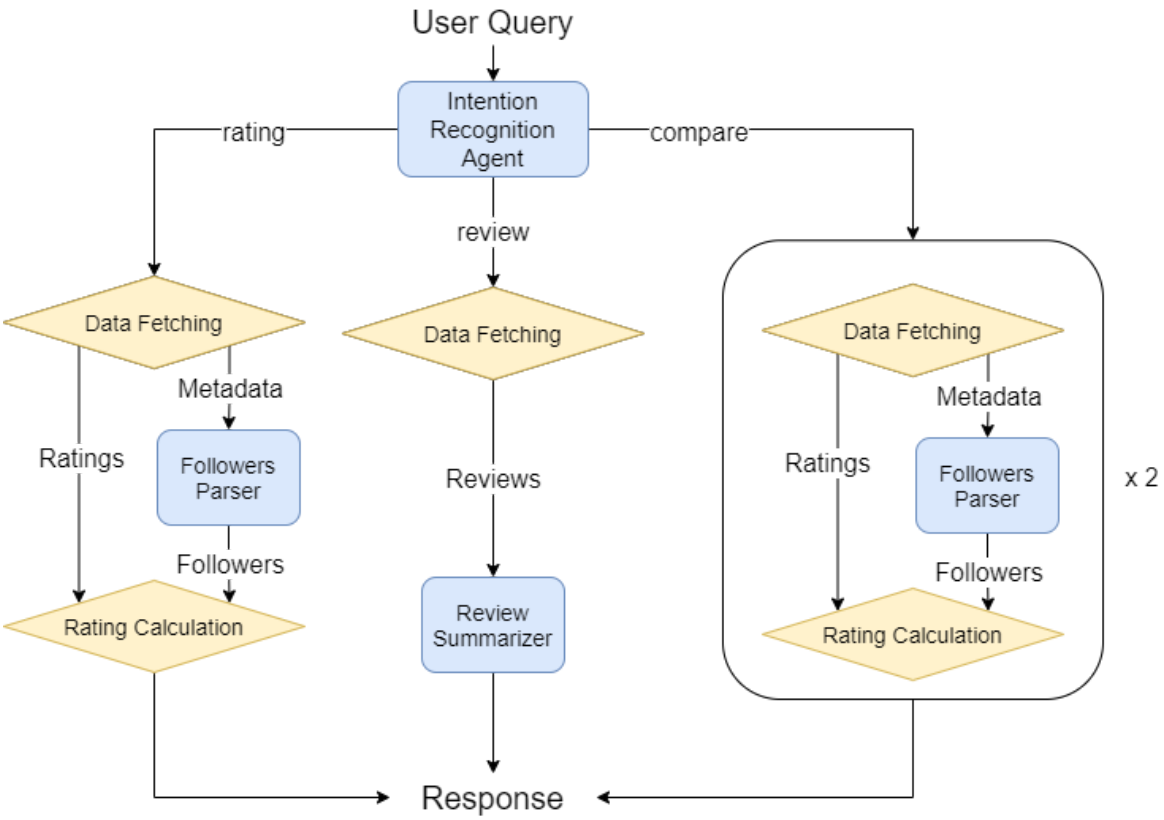
```
USER(Enter your query or EXIT to exit): Tell me something about The Tilt Bar Republic.
SYSTEM: The reviews for Tilt Republic Bar reflect a mixed sentiment, with some praising the extensive beer selection and good ambiance,
while others criticize the food quality, service issues, and overall value for money.
USER(Enter your query or EXIT to exit): How do people rate The Tilt Bar Republic?
SYSTEM: The overall rating of The Tilt Bar Republic is 3.275.
USER(Enter your query or EXIT to exit): Olive Garden or KFC, which one is better?
SYSTEM: Olive Garden has a higher rating than KFC, so you should go for Olive Garden.
USER(Enter your query or EXIT to exit): exit
```

Dataset

The dataset `reviews.csv` contains restaurant reviews with the following columns:

- `Restaurant`: The name of the restaurant.
- `Reviewer`: Name of the reviewer.
- `Review`: Text of the review.
- `Rating`: Integer rating (1–5).
- `Metadata`: String containing review and follower counts (e.g., "1 Review , 2 Followers").

General Workflow



1. **Intention Recognition**: Use the `IntentionRecognitionAgent` to identify the user's query type (rating, compare, or review) and extract relevant arguments (e.g., restaurant names).
2. **Data Fetching**: Fetch restaurant data from the CSV file using `fetch_restaurant_data`.
3. **Processing**: Process the data using the appropriate approach:
    - For rating queries, use `FollowersParserAgent` to extract follower counts from metadata and compute the weighted average rating using `calculate_overall_rating`.
    - For comparison queries, compute ratings for both restaurants and compare them.
    - For review queries, use `ReviewSummaryAgent` to summarize the reviews of the specified restaurant.

Task 1: Basic Functions

Implement the following functions in `main.py`:

- `fetch_restaurant_data`: Fetch restaurant data from the CSV file.
- `calculate_overall_rating`: Calculate the overall rating of a restaurant based on its reviews and follower counts.

Task 2: LLM Agent Implementation

Implement the following LLM agents in `main.py`:

- `IntentionRecognitionAgent`: Recognize the intention of the user query and extract relevant arguments.
- `FollowersParserAgent`: Parse the metadata to extract follower counts.
- `ReviewSummaryAgent`: Summarize the reviews of a restaurant.

For each agent, you will need to implement the `generate_response` method, which returns the generated response. You may need to parse the original LLM response to a structured format.

Setup

Environment Setup

Here are the steps to set up your environment for the project:

1. Create a new environment using conda or your preferred method:

```
conda create -n <your_env_name> python=3.10
conda activate <your_env_name>
```

2. Install the required packages:

```
pip install -r requirements.txt
```

Getting Started with HKUST GenAI

To use the HKUST GenAI API, you need to set up your API key. Follow these steps:

1. Please refer to the AZURE OPENAI API SERVICE documentation to generate your own API key. Please note that the API key should be kept confidential.
2. Fill in the `api_key` field in `api_key.json` with your API key. Then you can use your API key in the code to access the HKUST GenAI API.
3. We use `gpt-4o-mini` as the default model, and the cost for the whole task is usually less than 5 HKD, so the monthly free credit of HKD 8.00 should be enough for this project. You can check the credit usage at HKUST GenAI.

Testing, Submission and Grading

Testing

> For testing purposes, please do not modify `test.py`

The testing process for your implementation is divided into 2 parts:

- public test
- response generation

Public Test:

The public test is a set of queries to evaluate your implementation, which will not be counted for the final grades. Run it using:

```
python test.py
```

We provide a slight tolerance for each test case to account for any minor hallucinations.

If all the implementations are correct, you should see the following output:

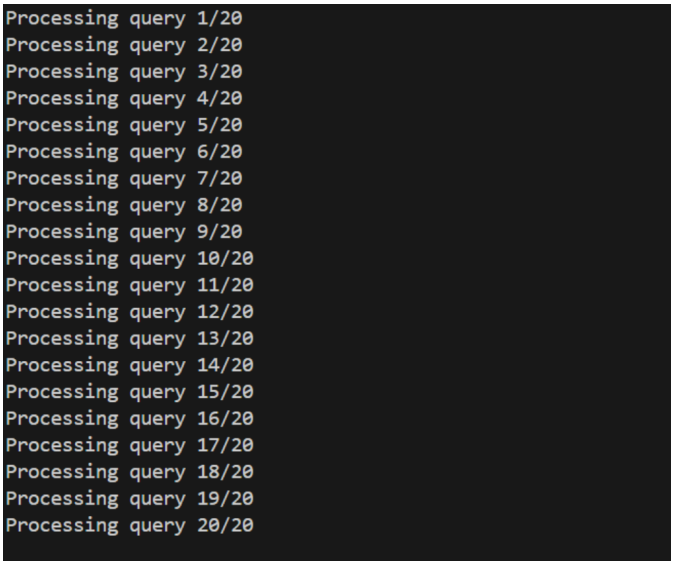
```
Test 1 Passed. Expected:  3.457 Query:  What's the user rating for SKYHY?
Test 2 Passed. Expected:  SKYHY Query:  Which is the better pick, Olive Garden or SKYHY?
Test 3 Passed. Expected:  3.897 Query:  What's the rating of Paradise?
Test 4 Passed. Expected:  SKYHY Query:  Should I choose Udipi's Upahar or SKYHY?
4/4 Tests Passed
```

Response Generation:

A script is provided to generate a predictions.csv file containing responses for a set of test queries. Run it using:

```
python test.py --generate
```

This will generate a `predictions.csv` file in the current directory, and the generation process will be logged in the console. As shown in the image below.



The `predictions.csv` file will be used for grading, which includes the following columns:

- `Query`: The user query we provide.
- `Response`: The generated response.

If the automatic generation produces invalid responses (e.g., incorrect formats at intermediate steps) due to hallucinations, you can run interactive queries and manually fill predictions.csv using the following workflow (The queries we used are in `predictions.csv`):

1. Run your program:

```
python main.py
```

1. Input your query in the console.
2. Obtain the query result from the console.
3. Manually paste the result into the appropriate cell in `predictions.csv`.

Please ensure the CSV file can be loaded correctly with `pandas` for grading.

Submission

Submit the following files:

- `main.py`: Your implementation of the LLM agents and functions.
- `predictions.csv`: The generated responses for the test queries.
- `report.pdf`: A brief report (1-3 pages) describing:
  - Your implementation approach.
  - The parsing method you used.
  - Results of your implementation.

DDL: May 11, 2025

Submission: Each team leader is required to submit the groupName.zip file that contains the report, predictions.csv, and main.py on canvas.

We will check your report with your code and the predictions.csv file. You will be graded based on the correctness of your implementation and your report.

Grading

- **Correctness (80%)**: Evaluated based on the correctness of responses in `predictions.csv`, we provide a slight tolerance for each test case to account for any minor hallucinations.
  - For each test case, we will extract the expected output from the response in `predictions.csv` and compare it with the expected output.
    - **Rating Queries**: A rating is considered correct if it is within $expected\_rating \pm 0.2$.
    - **Comparison Queries**: We will check whether the restaurant with the higher rating is correctly identified.
    - **Review Queries**: Any valid summary is acceptable.
- **Report (20%)**: Evaluated based on clarity, completeness.