Start-Ups Q&A AI Chatbot

Description

You are going to create a product for venture capital associates: an AI chatbot specialized in startup knowledge. It will respond to user questions by first searching for relevant information using neural search. Then, it will enhance the conversation by using a Large Language Model (LLM) to provide accurate and contextually appropriate responses.

Setup

- Development Environment: You should set up your development environment using your preferred language and framework. We recommend using Python, however it is not a requirement.
- Dependencies: You are free to use any language and framework. Please specify any necessary dependencies in your submission.
- We provide:
    - OpenAI's API key.
    - A basic LLM prompt for chat completion.

Assignment

Please read the entire assignment before starting 🙂

Task 1 - Create a Simple Neural Search Service

Your first task is to create a simple neural search service, using the following guide:
https://qdrant.tech/documentation/tutorials/neural-search/

This service should be used in the next task.

Once this task is done, 'GET http://localhost:8000/api/search?q=<query>' will return an answer.
For example: 'GET http://localhost:8000/api/search?q=Are there startups about wine?'

Task 2 - Create a Chatbot

Your second task is to create a chatbot that provides information and answers questions.

You will do that by creating a service that uses the neural search service you created in task 1 to query data, and then uses that data and OpenAI's chat completion API to return AI-generated responses.

Notes:

1. The service should implement a single endpoint named 'query' with the following payload:
    a. user_id: Unique identifier for the session (string)
    b. message: A message sent by the user (string)
2. The response should be in json format and the following payload:
    a. output: <AI-generated response> (string)
3. While using OpenAI's API, use one of the ChatGPT3.5-turbo models.
4. There is no requirement for addressing potential attacks on large language models, such as prompt injection or other malicious intents.
5. There is no requirement for UI.
6. There is no requirement for any data storage.

Task 3 - Add Context Support

Your third task is to add context support. Currently, each message is handled in a vacuum. However, a message can be a follow-up message to the previous one.

Take a close look at the following example:

User: "Are there startups about wine in Chicago?"
Chatbot: "Yes, 'Winestyr' in Chicago is a …"
→ User: "And in NY?" ←
Chatbot: "Absolutely! There is 'Wine for the World' in NY, …"

Every solution for handling the history is acceptable here. Using a DB is allowed but not required.

Task 4 - Managing Sequential Messages

Your fourth objective is to handle situations where users send multiple messages in quick succession. Consider this scenario:

1. A user initially asks, "Are there startups about wine in Chicago?"
2. Before a response is generated, the user sends a follow-up message, correcting the previous one: "Oops, I meant in New York!"

In such cases, the response should address the revised question: "Are there startups about wine in New York?"

Notes:

- This scenario involves two or more messages and thus two or more requests. At least one of these requests should be answered with the expected output, while the others may be answered with a unique response indicating the situation.
- Assume that users will not send more than 5 messages in a row.
- There is no requirement for rate limitation.

Provided Data:

You will be provided with the following files:

- prompt.txt - a file containing an LLM prompt template. This template can be used as the “system message” when using OpenAI’s Chat completion.

Requirements:

- Functionality:

        - Implement an endpoint named 'query' that receives POST requests with JSON payloads as defined above.
            - Use the retrieval service to fetch data
            - Use the OpenAI API to generate responses based on the received messages & fetched data
        - Ensure proper error handling and return appropriate error responses if necessary.
- Ensure data security and privacy by handling sensitive information appropriately (e.g., securely storing API keys).

Evaluation Criteria:

You will be evaluated based on the following criteria:

- Correctness and completeness of implementation.
- Code quality, including readability, maintainability, and adherence to best practices.
- Efficiency and performance of the backend service.
- Handling of error conditions.
- Adherence to provided specifications and guidelines.

Submission Instructions:

You should submit your completed assignments in 48 hours via GitHub repository. Please include all source code, configuration files, requirements, and any necessary documentation. Make sure the readme file contains all steps necessary to run the project.

The repo should be private and the following users should have access:

- yuval.cantor@paradox.ai
- inbar.halevi@paradox.ai
- yuval.berger@paradox.ai
- almog.idisis@paradox.ai
- alon.slutzky@paradox.ai

Additional Notes:

- API documentation for the OpenAI API can be found here.
- For questions or clarifications regarding the assignment, you can reach out to:
    - Yuval Cantor - 0506915967
    - Almog Idisis - 0528766440