MPCS Student Course Advisor

Final Project for Generative AI class

Muhammad Aulia Firmansyah

mauliafirmansyah@uchicago.edu

Purpose

- Experimenting with Generative AI Tools:
  - Langchain agent
  - Agent tool
  - RAG
  - Custom Document Loader
  - Memory
- Creating a usable product of Generative AI using Streamlit

Overview

- A chatbot to assist MPCS student with choosing and learning about the MPCS courses. This chatbot is built from LangChain agent.
- To fulfill its tasks, this chatbot uses 2 kinds of agent tools:
  - Scraper Agent Tool: Obtain course information from directly scraping the MPCS courses webpage based on specific input, such as year, quarter.
  - RAG Agent Tool: Retrieve other general information about MPCS program from external and internal website, which are previously loaded and stored in a vector store.
- To maintain correctness of the result, the chatbot is tasked to provide URL source in every response.

LangChain Agent

- LangChain agent is preferred over GPT because it doesn’t require ChatGPT-4 subscription. It currently use also has the potential to be LLM model agnostic.
- This tool is also chosen because of the ability to run tools. In each step, the agent decide which tool to use before composing a response.

Scraper Agent Tool

- This agent tool crawl directly to webpage using specific input, then scrape the web html content.
- To optimize the content size, the agent tool convert them to markdown format. This can reduce the content size to the 10-40% of the original size, removing all unnecessary html, css, and js code.
- The content then is returned to the agent for further action, attached with its request url for reference.
- We did not use RAG for this tool because we don’t know the number of actual webpages. Also, some webpages including course information may update frequently.

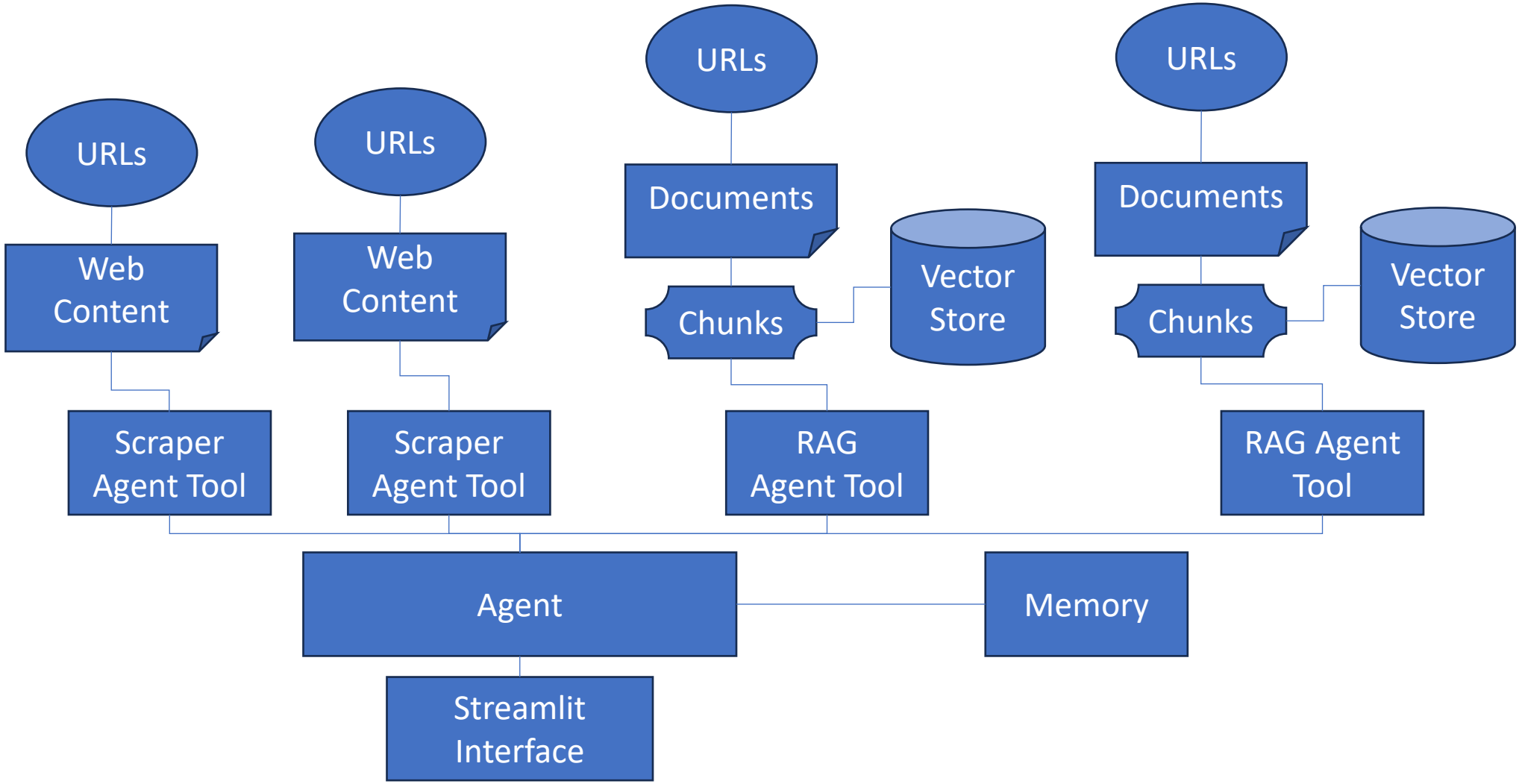
RAG Agent Tool

- Similar to scraper agent tool, this agent tool crawl to webpage.
- The difference is that this agent tool stores all webpages into vector store.
- The webpage is converted into document by using a custom loader which utilize the html to markdown function, to optimize the content size, while its url is saved as the metadata.
- In the retrieval process, the system perform similarity search based on query. Then, the topmost results are returned along with their url to the agent.

Memory

- The agent uses Conversation Summary Buffer to store memory.
- This memory combines the two ideas of buffering and summarization.
- It keeps a buffer of recent interactions in memory, but rather than just completely flushing old interactions it compiles them into a summary and uses both.
- It uses token length rather than number of interactions to determine when to flush interactions.

Complete Design



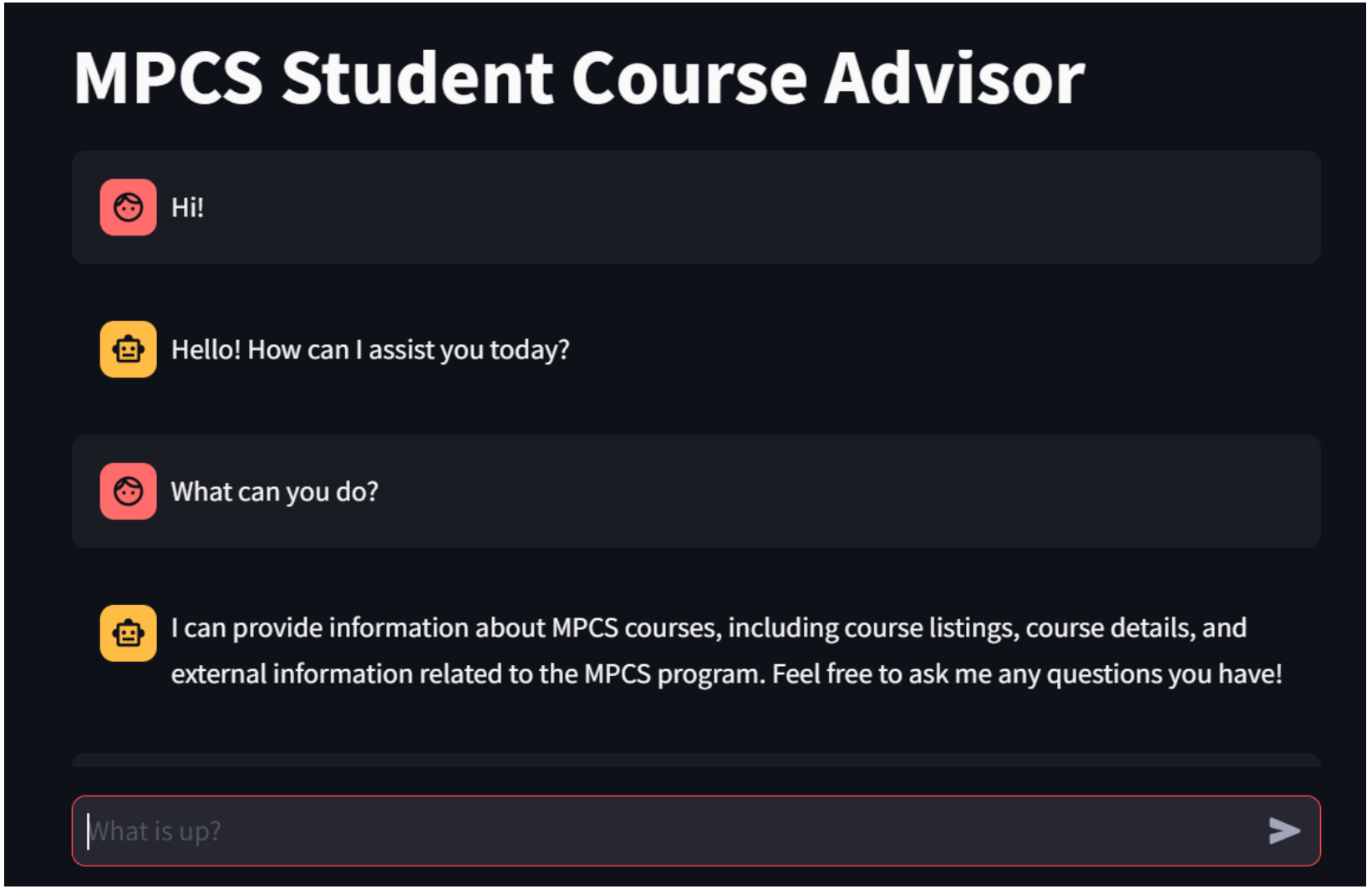
Running

- To run this project, just run:

```
streamlit run streamlit_app.py
```

- It will load RAG for first time and just load from the local vector store for the subsequent time, decreasing load time.

App Screenshot



Observation

- The chatbot can provide an answer which is faithful to the source, although it hallucinate sometimes.
- The results of running the same query may be different even though temperature is set to 0.
- There are times when an agent give an error when trying to run a tool. This is currently known issue in Langchain. (https://github.com/langchain-ai/langchain/issues/1559)
- Because RAG only provide chunks of documents from using similarity search, the inference might not be optimal if we answer have to be inferred from a lot of chunks.