💻 Technical Details

:AI: Structure of our Projects

:edge2: WebScraper

We employ Selenium and BeautifulSoup to extract both the dynamic content (generated through JavaScript) and the static content (HTML) from websites, facilitating comprehensive web scraping and data extraction tasks. The extracted information will be stored in vector database and to be retreived based on users question.

📲 Retrieval Augmented Generation

We use Retrieval Augmented Generation(RAG) to help LLM continuously learn any new knowledge so that we don't have to wait for a large amount of time to directly train it on these new knowledge. The specific steps of RAG include:

- Splitting the raw documents into smaller chunks
- Converting them into embeddings
- Storing those embeddings into a database
- Retrieving the relevant documents according to the query

Different retrieving polices can be used in this process, which have different performance on the accuracy and relevance of retrieved context, influencing the final answer generated by LLM. We are studying five different retrievers:

- **VectorstoreRetriever**: It's a lightweight wrapper around the vector store class to make it conform to the retriever interface. This retriever uses similarity search or maximum marginal relevance search based on the comparison of embedding distances.
- **MultiQuery**: Sometimes different results will be retrieved for a small change in the query when the embeddings didn't capture the semantics of the data well. MultiQuery retriever uses LLM to generate multiple queries from different perspectives for a given user input query, and for each of them, it retrieves a set of relevant documents and takes the unique union across all queries to get a larger set of potentially relevant documents.
- **Context compression**: If the retrieved document contains a lot of irrelevant information to the query, passing that full document can lead to more expensive LLM calls and poorer responses. Therefore, we can compress them using the context of the given query, so that only the relevant information is returned.
- **MultiVector**: This retriever can store multiple vectors for each document to better capture different semantic information that a document may have. It can either create a summary for each document or create hypothetical questions that each document would be appropriate to answer, and store them with that document.
- **Parent Document Retriever**: Smaller documents help embeddings better capture their semantics while larger documents should be returned to provide more complete context information. Parent Document Retriever first fetches the small chunks but then looks up the parent ids for those chunks and returns those larger documents.

:gpt: ChatBot

Our chatbot is powered by GPT-4. It can engage in multi-turn conversations with users based on the web content provided by them, addressing their real-world questions. It is capable of accepting multimodal inputs (such as images), and interacting with users based on these inputs, such as navigating based on a map image or recognizing text in an image to answer questions and other functions.

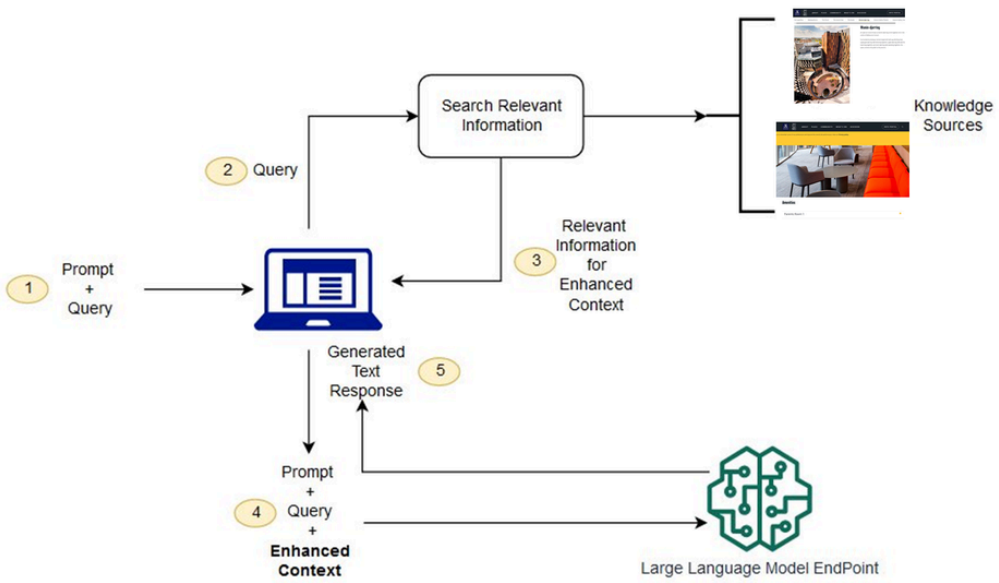
💻 Development Environment

Environment	Notes
Confluence	Office software
Jira	Office software
Trello	Office software
Github	Software Management
Python	Programming Language
Langchain	Python Library for RAG development
OpenAI	Python Library for GPT API

💻 Technology Stack

Python + Langchain + OpenAI

🏗 Architecture of the project



🐦 Useful Links

1.Course of LangChain - 60mins

LangChain for LLM Application Development - DeepLearning.AI

2.LangChain Official Docs

Introduction | 🦜🔗 Langchain

3.Different retrieve method

Retrievers | 🦜🔗 Langchain