DOCUMENTATION

Title: RAG-based Chatbot using LLM and VectorDB

This project implements a RAG-based chatbot for information retrieval using large language models(LLM), langchain and vectorDB set-up on a local environment using ollama. Chainlit is utilized for chatbot interaction

Main Components:

- **RetrievalQA (RAG Framework):**
  - Utilized for setting up a question-answering model based on the RAG framework, configured with the language model (LLM) and a vector store (Chroma).
  - RetrievalQA and RetrievalQAWithSourcesChain are part of the RAG (Retrieval Augmented Generation) framework.
  - The RetrievalQA.from_chain_type method is used to set up a RetrievalQA model.

- **Embedding:**
  - GPT4AllEmbeddings is used for text embeddings.
  - In both load.py and app.py, it is used to define the embedding function for processing text chunks.

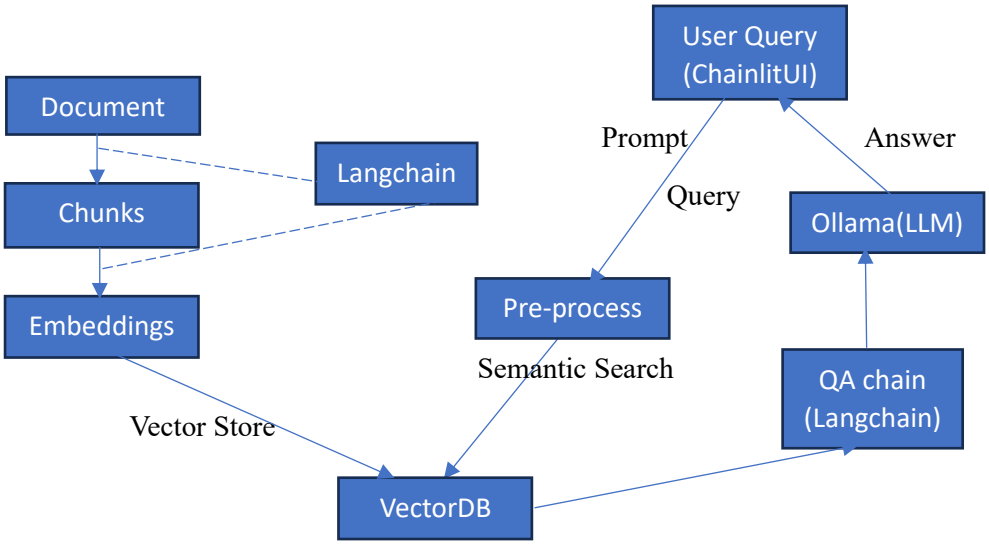
- **Langchain – Application framework for LLM**
  - Provides tools for interacting with language models, including retrieval-based models like RAG.
  - Facilitates the loading and management of models, embeddings, and other language resources.
  - Supports streaming output during asynchronous interactions.
  - Collaborates seamlessly with chainlit for chatbot development.

- **VectorDB (Chroma):**
  - Used for creating and persisting a Vector Database, serving as a retriever for the RAG model.

- **Ollama (LLM Framework):**
  - Ollama is used to load and manage the language model (LLM).
  - The specific model to be loaded, in this case, "mistral."

- **Chainlit:**
  - Simplifies asynchronous chatbot development.
  - Key Components:
    @cl.on_chat_start, @cl.on_message: Decorators for handling chat events.
    cl.user_session: Manages user session data.
  - Defines chat start and message handling functions, manages user sessions.

System Architecture:



Process:

First, langchain is used to read document. Then, the text-splitter is utilized to break it into chunks which are then processed through text embeddings (GPT4ALL) converting text data into vectors and these vectors are saved into vector database (Chroma). This allows us to process any document confidentially and locally.

Now in the QA process, when the user visits the chatbot using chainlit UI where bot displays message for user to ask query. Then, the query asked by user get pre-processed involving word embeddings to generate one or more vectors which are then used for semantic search in the vectorDB (similarity search). The search results are then passed onto LLMs (utilized locally using Ollama) for further processing to generate personalized responses. This process is integrated using Langchain framework. In this case a predefined QA chain is used for conversation (RetrievalQA chain). So, here langchain handles the backend logic and chainlit acts as an interaction logic.

Env Config:

- Linux : Kali
- Python3
- Ollama
- LLM Model: Mistral

Set up:

1. Go to the directory in which you want to work and Create and activate a virtual environment:

   ```
   python3 -m venv venv
   source venv/bin/activate
   ```

2. Install the required dependencies:

   ```
   pip install langchain langchain_community chainlit
   ```

3. Download Ollama for Linux using command:

   ```
   curl -fsSL https://ollama.com/install.sh | sh
   ```

   Start ollama for interaction using command:

   ```
   ollama serve
   ```

4. Paralelly download the model to be used (in this case mistral) using command:

   ```
   ollama run mistral
   ```

5. Run the load.py script to load documents and create a Vector Database (vdb):
   ```
   python3 load.py
   ```
6. Run the app.py script to start the chatbot:
   ```
   chainlit run app.py
   ```
7. Upon running app.py, a link to chainlit app is created. After opening the link, the chatbot will start and prompt you with a welcome message. You can interact with it by sending queries.