Project Overview

Estimated duration: 7 minutes

The final project of this course encourages you to use the tools and concepts you've learned so far. As you complete the project, you will need to upload screenshots of your responses to verify what you have done so that they can be graded in the peer-graded assignment.

Project overview: Build a QA Bot Web App

The project "Build a QA Bot Web App" is designed to integrate and implement the key concepts and tools you have learned in this course.

It simulates a real-world scenario where you are required to build a bot that will leverage LangChain and a large language model (LLM) to answer questions based on content from loaded PDF documents.

The project involves several key components and techniques including:

1. Load document using LangChain for different sources

Implement the `document_loader` function using `PyPDFLoader` from the `langchain_community` library to load PDF files, and capture a screenshot of your completed code.

Note: Capture the screenshot as `pdf_loader.png` after implementing the code.

2. Splitting long documents using text splitters

Complete the `text_splitter` function using `RecursiveCharacterTextSplitter` to split the loaded PDF content into manageable text chunks, and capture a screenshot of your completed code.

Note: Capture the screenshot as `code_splitter.png` after implementing the function.

3. Generating embeddings using embedding models

Complete the `watsonx_embedding()` function using the `WatsonxEmbeddings` class from the `langchain_ibm` library to generate text embeddings, and capture a screenshot of your completed code.

Note: Store the screenshot as `embedding.png` after implementing the function.

4. Storing embeddings using vector databases

Complete the `vector_database()` function to embed the text chunks using the `watsonx_embedding()` model and store them in a Chroma vector store using `Chroma.from_documents()`.

Note: Capture a screenshot of your completed code and save as `vectordb.png`.

5. Defining retrievers

Complete the `retriever(file)` function to load, split, embed, and convert documents into a retriever using similarity search from a Chroma vector store. Capture a screenshot of your completed code.

Note: Store the screenshot as `retriever.pnga` after implementing the function.

6. Setting up Gradio as the front-end interface

Define the `retriever_qa(file, query)` function using the RetrievalQA chain from langchain to perform question-answering over documents using RAG (Retrieval-Augmented Generation). Use `get_llm()` and `retriever()` in your implementation. Then connect this logic to a Gradio interface for interactive use.

Note: Store the screenshot as `QA_bot.png` showing:

- The Gradio interface created using `gr.Interface`
- A PDF uploaded to the interface

- The following user query entered in the textbox:

```
query = "What this paper is talking about?"
```

Through this project, you will gain hands-on experience in building an AI application using RAG and LangChain.