QA System Project

January 13, 2025

Contents

1 Introduction

This project implements a Question-Answering (QA) system leveraging the LangChain framework and OpenAI's APIs. The primary functionalities include loading PDF documents, splitting them into manageable chunks, storing embeddings in a vector database, and enabling conversational interactions with a retrieval-based memory.

2 Dependencies

The project requires the following Python libraries:

- `os`
- `openai`
- `langchain_community`
- `langchain`
- `langchain_openai`
- `dotenv`
- `shutil`

3 Setup and Configuration

The environment variable `OPENAI_API_KEY` must be set for accessing OpenAI's services. This can be configured using a `.env` file.

3.1 Environment Configuration

```
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv())
openai.api_key = os.environ['OPENAI_API_KEY']
```

4 Code Structure

The project consists of the following main components:

4.1 Document Loading

```
from langchain_community.document_loaders import PyPDFLoader

loaders = [PyPDFLoader(pdf_path)]
docs = []
for loader in loaders:
    docs.extend(loader.load())
```

4.2 Document Splitting

The documents are split into smaller chunks using `RecursiveCharacterTextSplitter` for better processing.

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

r_splitter = RecursiveCharacterTextSplitter(
    chunk_size=500,
    chunk_overlap=75,
    separators=["\n\n", "\n", r"(?<=\.␣)", "␣", ""]
)
splits = r_splitter.split_documents(docs)
```

4.3 Embedding and Vector Storage

Embeddings are generated and stored in a persistent Chroma database.

```
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
import shutil

embedding = OpenAIEmbeddings()
persist_directory = ’docs/chroma/’
if os.path.exists(persist_directory):
    shutil.rmtree(persist_directory)

vectordb = Chroma.from_documents(
    documents=splits,
    embedding=embedding,
    persist_directory=persist_directory
)
```

4.4 Retriever and Compression

An LLM-based compressor enhances the retriever's efficiency.

```
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import
    LLMChainExtractor
from langchain_openai import OpenAI

llm = OpenAI(temperature=0, model="gpt-3.5-turbo-instruct")
compressor = LLMChainExtractor.from_llm(llm)

compression_retriever = ContextualCompressionRetriever(
    base_compressor=compressor,
    base_retriever=vectordb.as_retriever(search_type="mmr")
)
```

4.5 Conversational QA System

The QA system integrates conversational memory using `ConversationBufferMemory`.

```
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationalRetrievalChain
from langchain_openai import ChatOpenAI

memory = ConversationBufferMemory(
    memory_key="chat_history",
    return_messages=True
)

llm01 = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)

qa = ConversationalRetrievalChain.from_llm(
    llm01,
    retriever=compression_retriever,
    memory=memory
)
```

5 Usage

To initialize the QA system, call the function `initialize_qa_system` with the path to the desired PDF file:

```
qa_system = initialize_qa_system("path/to/pdf")
```

6 Conclusion

This project demonstrates the integration of OpenAI's GPT model with LangChain to create a robust QA system capable of processing and retrieving information from large PDF documents.