Aim

To develop a Retrieval-Augmented Generation (RAG) system utilizing LLAMA3 for answering questions based on provided documents. The system aims to integrate various components for document ingestion, text splitting, vector embeddings, and retrieval, and provide a user-friendly interface using Streamlit.

Objectives

1. Integrate NVIDIAEmbeddings for generating document embeddings.
2. Use Streamlit to create a user interface for document embedding and question answering.
3. Implement document loading and splitting using Langchain components.
4. Store and retrieve document vectors using FAISS.
5. Generate accurate responses to user questions based on the context of provided documents using LLAMA3.

Summary

The project demonstrates the implementation of a RAG system leveraging LLAMA3 and various Langchain components. It includes loading documents, generating embeddings, splitting text into chunks, storing vectors in a FAISS vector store, and retrieving relevant documents to answer user queries. The interface is built using Streamlit, providing an interactive experience for embedding documents and querying them.

Tools and Libraries Used

- **Streamlit**: For creating the user interface.
- **Langchain**: For handling document processing and text splitting.
- **FAISS**: For storing and retrieving document vectors.
- **NVIDIA API**: For generating embeddings using NVIDIAEmbeddings.
- **PyPDF2**: For loading PDF documents.
- **dotenv**: For managing environment variables.

Procedure (Explain the Code)

1. **Import Libraries**:

CODE:

```
import streamlit as st

import os

from langchain_nvidia_ai_endpoints import NVIDIAEmbeddings, ChatNVIDIA

from langchain_community.document_loaders import WebBaseLoader, PyPDFDirectoryLoader
from langchain.embeddings import OllamaEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.chains.combine_documents import create_stuff_documents_chain
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain.chains import create_retrieval_chain
from langchain_community.vectorstores import FAISS
import time
from dotenv import load_dotenv
load_dotenv()
```

2. **Load Environment Variables**:

CODE:

```
os.environ['NVIDIA_API_KEY'] = os.getenv("NVIDIA_API_KEY")
```

3. **Vector Embedding Function**:

CODE:

```
def vector_embedding():
  if "vectors" not in st.session_state:
      st.session_state.embeddings = NVIDIAEmbeddings()
      st.session_state.loader = PyPDFDirectoryLoader("./data")
      st.session_state.docs = st.session_state.loader.load()
      st.session_state.text_splitter = RecursiveCharacterTextSplitter(chunk_size=700, chunk_overlap=50)
      st.session_state.final_documents =
st.session_state.text_splitter.split_documents(st.session_state.docs[:30])
      st.session_state.vectors = FAISS.from_documents(st.session_state.final_documents,
st.session_state.embeddings)
```

4. **Streamlit UI Elements**:

CODE:

```
st.title("Nvidia NIM Demo")
llm = ChatNVIDIA(model="meta/llama3-70b-instruct")
```

```
prompt = ChatPromptTemplate.from_template("""
Answer the questions based on the provided context only.
Please provide the most accurate response based on the question
<context>
{context}
<context>
Questions: {input}
""")

prompt1 = st.text_input("Enter Your Question From Documents")

if st.button("Documents Embedding"):
    vector_embedding()
    st.write("Vector Store DB Is Ready")
```

5. **Question Answering Logic**:

CODE:

```
if prompt1:
    document_chain = create_stuff_documents_chain(llm, prompt)
    retriever = st.session_state.vectors.as_retriever()
    retrieval_chain = create_retrieval_chain(retriever, document_chain)
    start = time.process_time()
    response = retrieval_chain.invoke({'input': prompt1})
    st.write(f"Response time: {time.process_time() - start}")
    st.write(response['answer'])

    with st.expander("Document Similarity Search"):
        for i, doc in enumerate(response["context"]):
            st.write(doc.page_content)
        st.write("--------------------------------")
```

Highlights

- **Integration of NVIDIAEmbeddings**: Utilizes NVIDIA's powerful embedding model for generating document embeddings.
- **Streamlit Interface**: Provides an interactive and user-friendly interface for embedding documents and querying them.
- **Efficient Document Processing**: Uses RecursiveCharacterTextSplitter for efficient text chunking and FAISS for vector storage and retrieval.
- **Responsive Question Answering**: Implements a retrieval chain using Langchain components to provide accurate and context-based answers.

Conclusion

The project successfully demonstrates the implementation of a RAG system using LLAMA3, NVIDIAEmbeddings, and Langchain components. The system can efficiently ingest documents, generate embeddings, and provide accurate answers to user queries based on the context of the provided documents. The use of Streamlit for the user interface ensures an interactive and seamless experience for users.