```
    Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3
    Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.11/di
    Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3
    Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/di
    Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/pyt
    Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/
    Requirement already satisfied: mypy-extensions>=0.3.0 in /usr/local/lib/p
    Requirement already satisfied: humanfriendly>=9.1 in /usr/local/lib/pytho
    Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pytho
```

```
!pip install pypdf
```

```
    Requirement already satisfied: pypdf in /usr/local/lib/python3.11/dist-pack
```

```
# Setting up Environment and Importing Libraries

import os
import bs4
import numpy as np
from langchain import hub
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import Chroma # A vector store used to hol
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough
from langchain_google_genai import ChatGoogleGenerativeAI, GoogleGenerativeAIEmb
from langchain.prompts import ChatPromptTemplate

# Set API Keys
os.environ['GOOGLE_API_KEY'] = "INSERT YOUR API KEY HERE"

#### INDEXING ####

# PDF Loader
from langchain_community.document_loaders import PyPDFLoader
loader = PyPDFLoader("/content/BRAT_wikipedia.pdf")
docs = loader.load()

# Split
text_splitter = RecursiveCharacterTextSplitter(chunk_size=3739, chunk_overlap=29
splits = text_splitter.split_documents(docs)

'''
used to split the loaded documents into smaller, manageable chunks (text) for pr
This is necessary as large documents (like PDFs) may be too long for models to h

chunk_size means that each chunk will be 900 characters long.
chunk_overlap means that there will be a 90-character overlap between adjacent c
'''
```

```
# Embed using Gemini API
embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")
vectorstore = Chroma.from_documents(documents=splits, embedding=embeddings)

'''
The embedding model (models/embedding-001) transforms the chunks of text into nu
Chroma is a vector store that stores these embeddings.
Chroma allows for efficient similarity-based retrieval later by indexing the vec

Chroma could be replaced by other vector stores like ANNOY, FAISS, Weaviate, or
'''

# Custom Cosine Similarity Retrieval
def cosine_similarity(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    return dot_product / (norm_vec1 * norm_vec2)

def custom_retrieve(query, k=5):
    query_embd = embeddings.embed_query(query)
    similarities = []

    for doc in splits:
        doc_embd = embeddings.embed_query(doc.page_content)
        sim = cosine_similarity(query_embd, doc_embd)
        similarities.append((sim, doc))

    # Sort documents by similarity score
    sorted_docs = sorted(similarities, key=lambda x: x[0], reverse=True)
    return [doc for _, doc in sorted_docs[:k]]  # Return top-k documents

'''
Alternative Retrieval Methods: Instead of custom cosine similarity,
you could use pre-built retrievers provided by LangChain, such as FAISS or Elast
'''

#### RETRIEVAL AND GENERATION####

# LLM (Gemini)
llm = ChatGoogleGenerativeAI(model="gemini-pro", temperature=0.7)

# Post-processing
def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)

# Retrieval and Question Handling
```

```
def retrieve_and_answer(question):
    # Retrieve top-k documents based on custom cosine similarity
    retrieved_docs = custom_retrieve(question)

    # Create the prompt with context
    context = format_docs(retrieved_docs)
    # print(context)
    template = """You are an assistant for question-answering tasks.
      Use the following pieces of retrieved context to answer the question.
      If you don't know the answer, just say that you don't know.
      Use three sentences maximum and keep the answer concise.
      Context: {context}
      Question: {question}
      Answer:
"""
    prompt_template = ChatPromptTemplate.from_template(template)
    # We are using the "rlm/rag-prompt" from langsmith
    # Chain with Gemini
    response = prompt_template | llm
    return response.invoke({"context": retrieved_docs, "question": question})

# Question
question = "Describe the music of BRAT."
response = retrieve_and_answer(question)

# Print the response
print(response)
```

```
WARNING:pypdf._reader:Ignoring wrong pointing object 186 0 (offset 0)
content='BRAT\'s music has been described variously by journalists as elect
```

Start coding or generate with AI.