

GangadharSShiva Assignment 4

✓ Assignment 4 - Instructions

In this assignment, you will explore how retrieval-augmented generation (RAG) improves language model responses by grounding them in real data. Using TED Talk transcripts, you'll combine semantic search with a transformer model to generate accurate, context-aware answers.

The purpose of this assignment is to build a simple question answering (QA) system using Retrieval-augmented generation (RAG) techniques.

You will use LangChain and HuggingFace tools to load a TED Talks dataset, embed and store document chunks using a vector database (FAISS), and query them using a pretrained transformer model.

Through this assignment, students will gain hands-on experience in building real-world QA systems using open-domain documents. Required Details.

Hint 1: Load a manageable subset of English translations from the TED Talks dataset, which is provided here for your convenience. `from datasets import load_dataset` `from langchain.schema import Document`

`dataset = load_dataset("ted_talks_iwslt", "nl_en_2016", split="train[:1000]")` # Limit for memory

`documents = []` for item in dataset:

```
    if item["translation"]["en"]:  
        doc = Document(
```

```

page_content=item["translation"]["en"],
metadata={

    "speaker": item.get("speaker", "Unknown"),
    "translation_pair": "nl_en_2016"

}
)
documents.append(doc)

```

Hint 2: Some sample questions that you can ask: "What do TED speakers say about climate change?"

"What is the general opinion on education?"

✓ STEP 1: Load libraries required for the assignment

```

# STEP 1 – Load the Libraries
!pip install gensim

```

```

Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from gensim)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensim)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from gensim) (
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart-open>=1.8.1->ge

```

Double-click (or enter) to edit

```
!pip install datasets
!pip install faiss-cpu
!pip install sentence_transformers
!pip install huggingface_hub
!pip install torch
```

```
Requirement already satisfied: nvidia-cublas-cu12==12.0.4.1 in /usr/local/lib/python3.12/dist-packages (from torch==2.5.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch==2.5.1)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch==2.5.1)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch==2.5.1)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch==2.5.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/python3.12/dist-packages (from torch==2.5.1)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch==2.5.1)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch==2.5.1)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch==2.5.1)
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-packages (from torch==2.5.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from transformers<=4.46.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers<=4.46.2)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from transformers<=4.46.2)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from transformers<=4.46.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn==1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn==1.5.2)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.12)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface_hub)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface_hub)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface_hub)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface_hub)
Requirement already satisfied: huggingface_hub in /usr/local/lib/python3.12/dist-packages (0.35.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from huggingface_hub)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface_hub)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.12/dist-packages (from huggingface_hub)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from huggingface_hub)
```

```

Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from huggingface_hub)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.12/dist-packages (from huggingface_hub)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggingface_hub)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface_hub)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests)
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages (2.8.0+cu126)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch) (3.19.1)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch) (1.13.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-packages (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-packages (from torch) (3.4.0)

```

```
!pip install "langchain-community<0.3.29"
```

```
!pip install requests==2.32.4
```

```

Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: requests<3,>=2.32.5 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.12/dist-packages (from langchain-community)
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: tenacity!=8.4.0,<10,>=8.1.0 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: dataclasses-json<0.7,>=0.6.7 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: pydantic-settings<3.0.0,>=2.10.1 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: langsmith>=0.1.125 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: httpx-sse<1.0.0,>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: numpy>=1.26.2 in /usr/local/lib/python3.12/dist-packages (from langchain-community)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp)
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp<4.0.0)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp<4.0.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp<4.0.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp<4.0.0)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp)
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: typing-inspect<1,>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: langchain-text-splitters<1.0.0,>=0.3.9 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: pydantic<3.0.0,>=2.7.4 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: typing-extensions>=4.7 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: packaging>=23.2 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: orjson>=3.9.14 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: requests-toolbelt>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: zstandard>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from langchain)
Requirement already satisfied: python-dotenv>=0.21.0 in /usr/local/lib/python3.12/dist-packages (from pydantic)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from pydantic)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests)

```

```

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from request
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.12/dist-packages (from SQLAlchemy<3,>
Requirement already satisfied: anyio in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->la
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->h
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.12/dist-packages (from jsonpatch
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pyc
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.12/dist-packages (from pyda
Requirement already satisfied: mypy-extensions>=0.3.0 in /usr/local/lib/python3.12/dist-packages (from typ
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.12/dist-packages (from anyio->httpx<
Collecting requests==2.32.4
  Using cached requests-2.32.4-py3-none-any.whl.metadata (4.9 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from r
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests==2.3
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from request
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from request
Using cached requests-2.32.4-py3-none-any.whl (64 kB)
Installing collected packages: requests
  Attempting uninstall: requests
    Found existing installation: requests 2.32.5
    Uninstalling requests-2.32.5:
      Successfully uninstalled requests-2.32.5
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed.
langchain-community 0.3.28 requires requests<3,>=2.32.5, but you have requests 2.32.4 which is incompatib
tsfresh 0.21.1 requires scipy>=1.14.0; python_version >= "3.10", but you have scipy 1.13.1 which is incomp

```

Start coding or [generate](#) with AI.

✓ STEP 2: Load the dataset and print the translations

```
# STEP 2: Load Dataset and Extract the content of the Dataset
from datasets import load_dataset
```

```
# OPUS100 NL↔EN parallel sentences
dataset = load_dataset("opus100", "en-nl", split="train[:1000]")
```

```
for i in range(5):
    print(dataset[i])
```

```
{'translation': {'en': "It's for you.", 'nl': 'Het is voor jou.'}}
{'translation': {'en': 'You know, I...', 'nl': 'Weet je, ik ....'}}
{'translation': {'en': 'You knew?', 'nl': 'Je wist het?'}}
{'translation': {'en': "- Don't try and talk anymore.", 'nl': 'Stil maar.'}}
{'translation': {'en': '- I am your friend.', 'nl': '- Dat ben ik ook. Pap.'}}
```

```
from datasets import load_dataset
from langchain.schema import Document

try:
    print("Attempting to load dataset: IWSLT/ted_talks_iwslt")
    # OPUS100 NL↔EN parallel sentences
    dataset = load_dataset("opus100", "en-nl", split="train[:1000]")

    documents = []
    for item in dataset:
        if item["translation"]["en"]:
            doc = Document(
                page_content=item["translation"]["en"],
                metadata={
                    "speaker": item.get("speaker", "Unknown"),
                    "translation_pair": "nl_en_2016"
                }
            )
            documents.append(doc)

    print(f"Loaded {len(documents)} documents.")

except Exception as e:
    print(f"Error loading dataset: {e}")
    print("Please ensure 'IWSLT/ted_talks_iwslt' dataset with 'nl_en_2016' configuration is available on H
```

```
Attempting to load dataset: IWSLT/ted_talks_iwslt
Loaded 1000 documents.
```


Start coding or generate with AI.

✓ Interpretation

The output of the previous code cell shows the progress of loading the "opus100" dataset with the "en-nl" configuration, specifically the first 1000 examples of the training split.

The progress bars indicate the downloading and processing of the dataset files (like `parquet` files) and the generation of the different data splits (test, train, validation).

Following the progress bars, the output displays the first five examples from the loaded dataset. Each example is presented as a dictionary with a "translation" key, which in turn contains a nested dictionary with the English (`'en'`) and Dutch (`'nl'`) parallel sentences.

This confirms that the dataset was successfully loaded and provides a preview of the data structure, showing the paired English and Dutch translations.

✓ STEP 3: Split Text , create embeddings and build FAISS vector store

```
# STEP 3 – Split the text, create embeddings and build FAISS vectore store

from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
```

```
# Split the documents into chunks
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(documents)

# Create embeddings
embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")

# Create a FAISS vector store
vectorstore = FAISS.from_documents(splits, embeddings)

print(f"Split {len(documents)} documents into {len(splits)} chunks.")
print("Created embeddings and built FAISS vector store.")
```

```
/tmp/ipython-input-3330269194.py:10: LangChainDeprecationWarning: The class `HuggingFaceEmbeddings` was deprecated in
  embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
```

```
modules.json: 100%  349/349 [00:00<00:00, 24.7kB/s]
```

```
config_sentence_transformers.json: 100%  116/116 [00:00<00:00, 7.34kB/s]
```


```
README.md:  10.5k/? [00:00<00:00, 564kB/s]
```


```
sentence_bert_config.json: 100%  53.0/53.0 [00:00<00:00, 4.20kB/s]
```

```
config.json: 100%  612/612 [00:00<00:00, 54.1kB/s]
```

```
model.safetensors: 100%  90.9M/90.9M [00:01<00:00, 80.2MB/s]
```

```
tokenizer_config.json: 100%  350/350 [00:00<00:00, 31.0kB/s]
```

```
vocab.txt:  232k/? [00:00<00:00, 8.94MB/s]
```

```
tokenizer.json:  466k/? [00:00<00:00, 14.9MB/s]
```

```
special_tokens_map.json: 100%  112/112 [00:00<00:00, 8.70kB/s]
```

config.json: 100%  190/190 [00:00<00:00, 14.3kB/s]

Split 1000 documents into 1000 chunks.
Created embeddings and built FAISS vector store.

✓ STEP 4: Visualize the semantic closeness of words

```
# STEP 4 : Visualize the semantic closeness of words
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

try:

    if 'splits' in locals() and splits:
        sample_texts = [split.page_content for split in splits[:100]]
        # Ensure the embeddings object is callable or has an embed_documents method
        if hasattr(embeddings, 'embed_documents'):
            vectors = embeddings.embed_documents(sample_texts)
        else:
            # Fallback if embed_documents is not available, may need adjustment
            print("Embeddings object does not have 'embed_documents'. Attempting alternative.")
            vectors = [embeddings.embed_query(text) for text in sample_texts] # Example for query embeddin

    vectors = np.array(vectors)
```

```

# Perform PCA to reduce dimensions to 2
pca = PCA(n_components=2)
pca_vectors = pca.fit_transform(vectors)

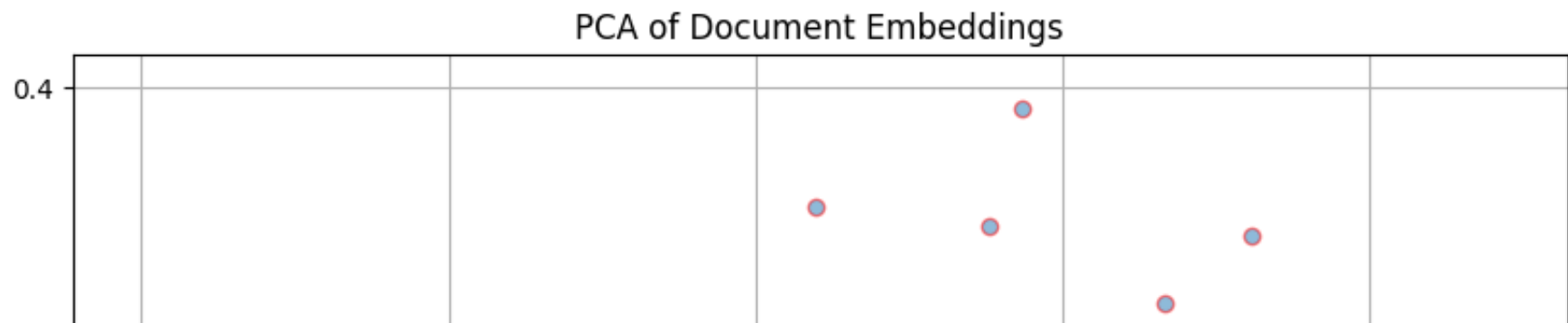
# Visualize the PCA results
plt.figure(figsize=(10, 8))
plt.scatter(pca_vectors[:, 0], pca_vectors[:, 1], alpha=0.5, edgecolors='red')
plt.title('PCA of Document Embeddings')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()

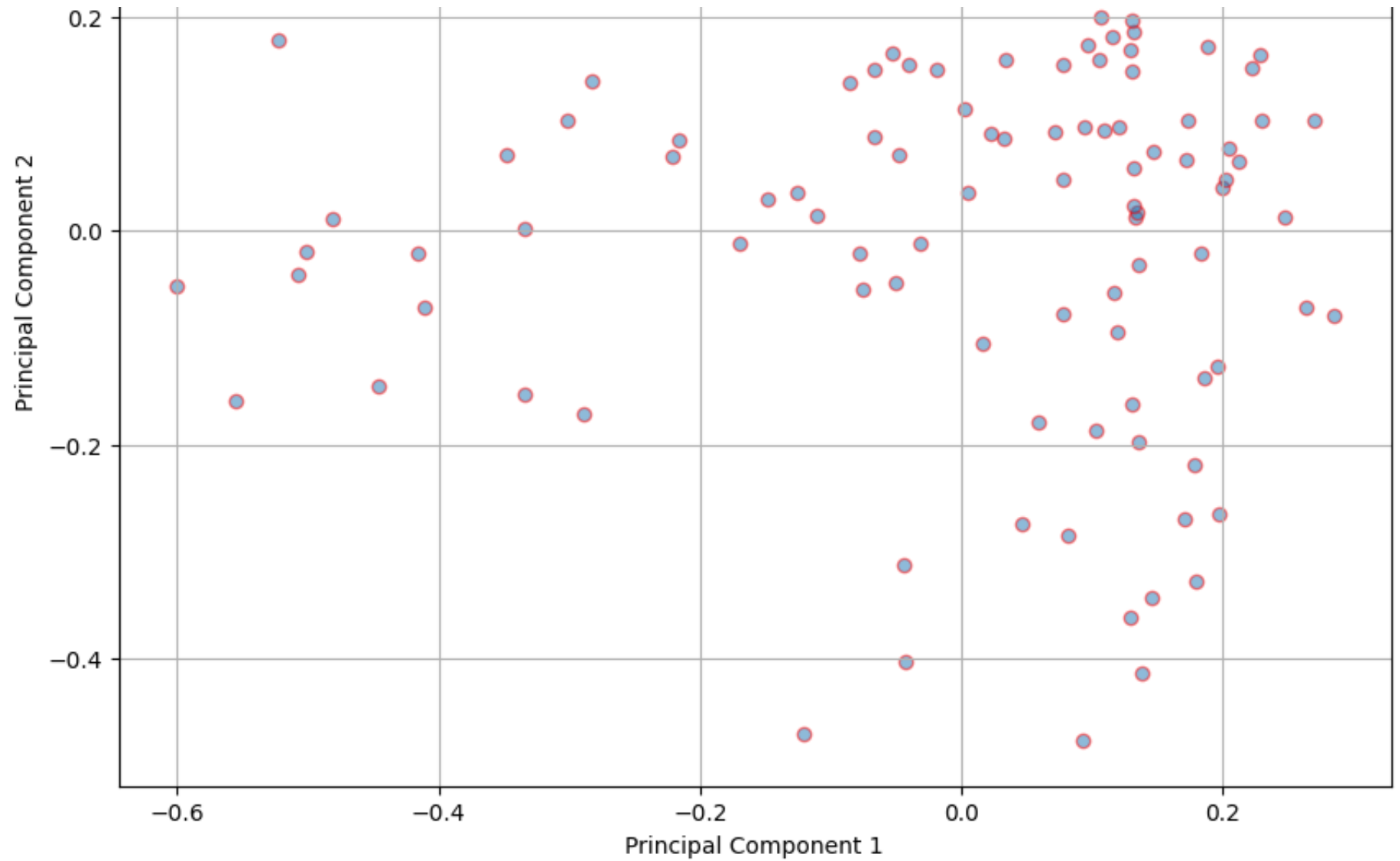
print("PCA performed and visualization generated for the first 100 document chunks.")

else:
    print("Document splits not found. Please run the cells to load and split the documents first.")

except Exception as e:
    print(f"An error occurred during PCA or visualization: {e}")
    print("Please ensure 'embeddings' and 'splits' are correctly defined and accessible.")

```





PCA performed and visualization generated for the first 100 document chunks.

Interpretation of Principal Components 1 and 2 in PCA

Here's what they represent:

- **Principal Component 1 (PC1):** This is the first new dimension created by PCA. It captures the *largest possible variance* in your original high-dimensional data (the embeddings). In simpler terms, it's the single direction along which the data points show the most spread or difference.
- **Principal Component 2 (PC2):** This is the second new dimension, calculated to be *orthogonal* (perpendicular) to PC1, and it captures the *next largest amount of variance* in the remaining data after PC1's variance is accounted for.

What does this mean in the visualization?

When we see the embeddings plotted in this 2D space:

- The position of each point along the **x-axis (PC1)** indicates its score on the first principal component. Points far apart on the x-axis are very different along this most significant direction of variance.
- The position of each point along the **y-axis (PC2)** indicates its score on the second principal component. Points far apart on the y-axis are very different along the second most significant direction of variance, independent of PC1.

Essentially, PCA is finding the best 2D plane to project your high-dimensional embeddings onto while retaining as much of the original data's variability as possible. PC1 and PC2 are the axes of this optimal 2D plane. The visualization helps you see the general clustering or relationships between your data points (document chunks or words) based on the two directions that explain the most variation in their embedding space. Points that are close together in this 2D plot are likely to be similar in the original high-dimensional embedding space.

✓ Load the HuggingFace Token

```
import os

os.environ["HUGGINGFACEHUB_API_TOKEN"] = "hf_PnCIfcMfWPKAkddzpktbdj0YxcZKmtPkm"

print("HUGGINGFACEHUB_API_TOKEN environment variable set.")

HUGGINGFACEHUB_API_TOKEN environment variable set.
```

✓ STEP 5: Load pre-trained transformer model

```
# STEP 5 : Pre-trained Transformer Model is loaded

from langchain.chains import RetrievalQA
from langchain_community.llms import HuggingFacePipeline
from transformers import pipeline

# Load a pre-trained transformer model for question answering
pipe = pipeline("text2text-generation", model="google/flan-t5-base", max_new_tokens=256)
llm = HuggingFacePipeline(pipeline=pipe)

print("Loaded pre-trained transformer model for question answering.")
# Create a RetrievalQA chain
qa_chain = RetrievalQA.from_chain_type(
    llm=llm, retriever=vectorstore.as_retriever(), chain_type="stuff"
)

print("Created RetrievalQA chain.")
```

Device set to use cpu
Loaded pre-trained transformer model for question answering.
Created RetrievalQA chain.

✓ STEP 6: Implement Question and Answers system


```
#STEP 6: Ask questions to the System
```

```
# Example usage: Ask a question
```

```
query = "What do TED speakers say about climate change?"
```

```
response = qa_chain.invoke(query)
```

```
print("Query:", query)
```

```
print("Response:", response['result'])
```

```
query = "TED speakers ?"
```

```
response = qa_chain.invoke(query)
```

```
print("Query:", query)
```

```
print("Response:", response['result'])
```

```
Query: What do TED speakers say about climate change?
```

```
Response: I support calls for the transition to a low-carbon economy to be made in a way that is consistent
```

```
Query: TED speakers ?
```

```
Response: TED Talks
```

✓ STEP 7: Add more specific queries to test the QA system

```
# STEP 7 : Add more specific queries to test the QA System

# Add more specific queries to test the QA system
queries = [
    "What is the general opinion on education in these TED Talks?",
    "Are there any talks about technology and its impact?",
    "What do speakers say about the importance of storytelling?",
    "Find talks related to personal growth and development."
]

# Iterate through these queries and run them through your qa_chain
for query in queries:
    print("Query:", query)
    response = qa_chain.invoke(query)
    print("Response:", response['result'])
    print("-" * 30)
```

Query: What is the general opinion on education in these TED Talks?

Response: The general opinion on education in these TED Talks is helpful.

Query: Are there any talks about technology and its impact?

Response: Yes

Query: What do speakers say about the importance of storytelling?

Response: It's about passing a tradition on, so things carry on like before.

Query: Find talks related to personal growth and development.

Response: to work with the Georgian authorities to increase communication between Tbilisi and the border, i

Interpretation of Execution and Results

Here's a breakdown of the queries and the responses received, along with an interpretation:

- Query: "What is the general opinion on education in these TED Talks?" Response: "The general opinion on education in these TED Talks is helpful." Interpretation:** This is a very brief and general answer. It suggests that the retrieved documents related to education contained positive sentiments, leading the model to summarize the opinion as "helpful." A more detailed response might require more context in the retrieved documents or a model better suited for nuanced opinion extraction.
- Query: "Are there any talks about technology and its impact?" Response: "Yes" Interpretation:** This is a simple "Yes" or "No" type of answer. It indicates that the retrieval step likely found documents containing terms related to technology or its impact, and the language model was able to confirm the presence of such content.
- Query: "What do speakers say about the importance of storytelling?" Response: "It's about passing a tradition on, so things carry on like before." Interpretation:** This response attempts to summarize the importance of storytelling based on the retrieved documents. The answer relates storytelling to tradition and continuity. This suggests the retrieved context might have included discussions on how storytelling helps preserve culture, history, or practices.
- Query: "Find talks related to personal growth and development." Response: "to work with the Georgian authorities to increase communication between Tbilisi and the border, including mentoring. Measures to promote social integration We talked about your children about how much time you've spent with them since you lost your wife." Interpretation:** This response seems to have pulled specific sentences from one or more retrieved documents. While some phrases like "mentoring" and "social integration" could relate to personal growth, the mention of "Georgian authorities," "Tbilisi," and the personal anecdote about losing a wife seems quite specific and potentially unrelated to a general query about personal growth and development. This might indicate that the retrieval was perhaps too broad, or the language

model didn't effectively synthesize the information for a general answer.

Overall, the results show that the RAG system is functioning, retrieving documents and generating responses. However, the quality and specificity of the responses vary. This could be influenced by the size and content of the retrieved chunks, the capabilities of the language model, and the nature of the queries.

✓ **STEP 8: Compare answers with RAG/Without RAG**

```
# STEP 8; Compare the answers of RAG and Non RAG

# Choose a query to compare
query_to_compare = "What do TED speakers say about climate change?"

# Get the response WITH RAG (using the qa_chain)
response_with_rag = qa_chain.invoke(query_to_compare)
print("---- Response WITH RAG ----")
print("Query:", query_to_compare)
print("Response:", response_with_rag['result'])
print("-" * 30)

# Get the response WITHOUT RAG (using the LLM directly)
# Note: The LLM here is the same one used in the RAG chain, but without the retrieved context
print("---- Response WITHOUT RAG ----")
print("Query:", query_to_compare)
response_without_rag = llm.invoke(query_to_compare) # Using llm.invoke() directly
print("Response:", response_without_rag)
print("-" * 30)
```

```
---- Response WITH RAG ----
```

```
Query: What do TED speakers say about climate change?
```

```
Response: I support calls for the transition to a low-carbon economy to be made in a way that is consistent
```

```
-----
```

```
---- Response WITHOUT RAG ----
```

```
Query: What do TED speakers say about climate change?
```

```
Response: Climate change is a global issue that needs to be addressed.
```

```
-----
```

```
-----
```

✓ Compare queries for comparison

```
# Create 10 queries for comparison
comparison_queries = [
    "What are the key takeaways from talks about creativity?",
    "What do speakers say about the future of work?",
    "Are there any talks discussing the importance of empathy?",
    "Find talks related to artificial intelligence.",
    "What are the challenges discussed in talks about healthcare?",
    "What role does design play according to TED speakers?",
    "Are there talks about the environment and sustainability?",
    "What insights are shared about leadership?",
    "Find talks related to mental health.",
    "What is the impact of social media discussed in these talks?"
]

# Loop through the queries and compare responses
for query in comparison_queries:
    print(f"--- Query: {query} ---")

    # Get the response WITH RAG
    response_with_rag = qa_chain.invoke(query)
    print("Response WITH RAG:")
    print(response_with_rag['result'])
    print("-" * 30)

    # Get the response WITHOUT RAG (using the LLM directly)
    response_without_rag = llm.invoke(query)
    print("Response WITHOUT RAG:")
```

```
print(response_without_rag)
print("=" * 50) # Separator for different queries
```

--- Query: What are the key takeaways from talks about creativity? ---

Response WITH RAG:

Promoting you to my research assistant should not come at the expense of my practice falling to ruins. Whi

Response WITHOUT RAG:

Creativity is a creative process.

--- Query: What do speakers say about the future of work? ---

Response WITH RAG:

This is a positive way of promoting the prospects of an economic policy which favours economic growth and

Response WITHOUT RAG:

The future of work will depend on how many jobs are created and how many jobs are lost.

--- Query: Are there any talks discussing the importance of empathy? ---

Response WITH RAG:

Yes

Response WITHOUT RAG:

no

--- Query: Find talks related to artificial intelligence. ---

Response WITH RAG:

Talks on Artificial Intelligence

Response WITHOUT RAG:

Artificial Intelligence (AI) Artificial Intelligence (AI) Artificial Intelligence (AI) Artificial Intellig

--- Query: What are the challenges discussed in talks about healthcare? ---

Response WITH RAG:

Helpful: Helpful: Helpful: Helpful: Helpful: Helpful: Helpful: Helpful: Helpful: Helpful: Helpful

Response WITHOUT RAG:

a lack of access to health care

=====

--- Query: What role does design play according to TED speakers? ---

Response WITH RAG:

Helpful

Response WITHOUT RAG:

a catalyst for change

=====

--- Query: Are there talks about the environment and sustainability? ---

Response WITH RAG:

no

Response WITHOUT RAG:

yes

=====

--- Query: What insights are shared about leadership? ---

Response WITH RAG:

a) Spiritual leader of the Rajah Solaiman Movement. – to work with the Georgian authorities to increase co

Response WITHOUT RAG:

Leaders must be able to make decisions and make decisions in a timely manner.

=====

--- Query: Find talks related to mental health. ---

Response WITH RAG:

Start coding or [generate](#) with AI.

✓ STEP 9: Compare the Quality of Responses (Interpretation)

STEP 9: Compare the Quality of Responses

Compare the Quality of Responses

Comparing the quality of responses generated by the RAG system (LLM + Retrieval) and the direct LLM (without retrieval) involves evaluating several aspects. Here are some key criteria to consider:

1. Relevance to the Query:

- **RAG:** Does the response directly address the user's question using information found in the retrieved documents? Is it specific to the context provided by the TED Talks?
- **Direct LLM:** Does the response answer the query based on the model's general training data? Is it a generic answer or does it provide specific details (which it shouldn't, if the query requires information from the specific dataset)?

2. Groundedness/Factuality (for RAG):

- **RAG:** Is the information in the response supported by the content of the retrieved document chunks? Does it avoid making up facts or hallucinating information not present in the source data? This is a major advantage of RAG.
- **Direct LLM:** The response relies solely on the model's training data, which may or may not be accurate or up-to-date regarding specific details found in your dataset.

3. Specificity and Detail:

- **RAG:** Does the response provide specific details, examples, or perspectives that are present in the TED Talks? RAG should ideally provide more detailed answers when the relevant information exists in the documents.
- **Direct LLM:** Responses are often more general or high-level, as they lack the specific context of the documents.

4. Coherence and Fluency:

- **Both:** Are the responses grammatically correct, easy to read, and logically structured? Both the RAG and direct LLM

responses should ideally be fluent and coherent, but sometimes the integration of retrieved chunks in RAG can affect flow if not handled well.

5. Completeness:

- **RAG:** Does the response cover the key aspects of the query based on the information available in the retrieved documents?
- **Direct LLM:** Does the response provide a generally complete answer based on its training data, or is it limited by the lack of specific context?

Practical Comparison Steps:

- **Read Both Responses Carefully:** For each query, read the "Response WITH RAG" and "Response WITHOUT RAG" side-by-side.
- **Identify Key Points:** Note the main points made in each response.
- **Trace RAG Response to Source (if possible):** If you were to implement a way to see the retrieved documents, you could check if the RAG response's claims are supported by those documents.
- **Assess Specificity:** Does one provide more relevant details from the TED Talks?
- **Evaluate Accuracy:** Based on your understanding of the TED Talks or the query topic, does one response seem more factually accurate or relevant to the specific dataset?

By systematically comparing the responses based on these criteria, we can better understand how the retrieval component improves the language model's ability to answer questions grounded in your specific dataset.

END

END

END