```
pip install langchain langchain-community langchain-huggingface sentence-transformers faiss-cpu transformers torch pypdf doc
```

```
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: huggingface_hub
    Found existing installation: huggingface-hub 0.28.1
    Uninstalling huggingface-hub-0.28.1:
      Successfully uninstalled huggingface-hub-0.28.1
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
  Attempting uninstall: langchain-core
    Found existing installation: langchain-core 0.3.40
    Uninstalling langchain-core-0.3.40:
      Successfully uninstalled langchain-core-0.3.40
  Attempting uninstall: langchain-text-splitters
    Found existing installation: langchain-text-splitters 0.3.6
    Uninstalling langchain-text-splitters-0.3.6:
      Successfully uninstalled langchain-text-splitters-0.3.6
  Attempting uninstall: langchain
    Found existing installation: langchain 0.3.19
    Uninstalling langchain-0.3.19:
      Successfully uninstalled langchain-0.3.19
Successfully installed dataclasses-json-0.6.7 docx2txt-0.9 faiss-cpu-1.11.0 hf-xet-1.1.0 httpx-sse-0.4.0 huggingface_h
```

```python
import os
import time
from langchain_community.document_loaders import DirectoryLoader, TextLoader, PyPDFLoader, Docx2txtLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import FAISS
from langchain_huggingface import HuggingFaceEmbeddings
from langchain.prompts import ChatPromptTemplate
from langchain_community.llms import HuggingFacePipeline
from langchain.schema.runnable import RunnablePassthrough
from langchain.schema.output_parser import StrOutputParser
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
from huggingface_hub import login
```

```python
# Step 1: Set Hugging Face token (optional for public models, required for gated models)
HF_TOKEN = "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"
if HF_TOKEN:
    login(token=HF_TOKEN)
else:
    print("Warning: HF_TOKEN not set. Public models will work, but gated models may fail.")
```

```
# Step 2: Define directory and load multiple document types
directory_path = "/content/drive/MyDrive/documents/"
def create_loader(file_path):
    if file_path.endswith(".txt"):
        return TextLoader(file_path, encoding="utf8")
    elif file_path.endswith(".pdf"):
        return PyPDFLoader(file_path)
    elif file_path.endswith(".docx"):
        return Docx2txtLoader(file_path)
    return None

print("🔁 Loading documents from", directory_path, "...")
documents = []
for file_name in os.listdir(directory_path):
    file_path = os.path.join(directory_path, file_name)
    loader = create_loader(file_path)
    if loader:
        try:
            documents.extend(loader.load())
        except Exception as e:
            print(f"Error loading {file_name}: {e}")
if not documents:
    raise ValueError("No valid documents found in the directory.")
```

```
🔁 Loading documents from /content/drive/MyDrive/documents/ ...
```

```
# Step 3: Split documents into chunks
text_splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
text_chunks = text_splitter.split_documents(documents)
print(f"Created {len(text_chunks)} document chunks.")
```

```
Created 529 document chunks.
```

```
# Step 4: Create vector embeddings using HuggingFace
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens),
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json:  100%                                    349/349  [00:00<00:00,  28.4kB/s]
config_sentence_transformers.json:  100%               116/116  [00:00<00:00,  7.95kB/s]
README.md:  100%                                       10.5k/10.5k  [00:00<00:00,  1.14MB/s]
sentence_bert_config.json:  100%                       53.0/53.0  [00:00<00:00,  6.02kB/s]
config.json:  100%                                     612/612  [00:00<00:00,  64.5kB/s]
model.safetensors:  100%                               90.9M/90.9M  [00:01<00:00,  108MB/s]
tokenizer_config.json:  100%                           350/350  [00:00<00:00,  33.1kB/s]
vocab.txt:  100%                                       232k/232k  [00:00<00:00,  3.38MB/s]
tokenizer.json:  100%                                  466k/466k  [00:00<00:00,  3.33MB/s]
special_tokens_map.json:  100%                         112/112  [00:00<00:00,  6.95kB/s]
config.json:  100%                                     190/190  [00:00<00:00,  19.7kB/s]
```

```
# Step 4: Create vector embeddings using HuggingFace
# embedding_model = HuggingFaceEmbeddings(model_name="meta-llama/Llama-2-7b-chat-hf")
```

```
# Step 5: Store in FAISS vector database and persist
faiss_index_path = "faiss_index"
if os.path.exists(faiss_index_path):
    print("🔁 Loading existing FAISS index...")
    vectorstore = FAISS.load_local(faiss_index_path, embedding_model, allow_dangerous_deserialization=True)
else:
    print("🔁 Creating new FAISS index...")
    vectorstore = FAISS.from_documents(text_chunks, embedding_model)
    vectorstore.save_local(faiss_index_path)
```

```
retriever = vectorstore.as_retriever(search_kwargs={"k": 3})
```

```
🔁 Creating new FAISS index...
```

```
# Step 6: Define Prompt Template
prompt_template = """
You are a precise question-answering assistant for a Retrieval-Augmented Generation system.
Answer the question based solely on the provided context, without using external knowledge.
If the context does not contain enough information to answer, respond with "I don't have enough information to answer."
Provide a direct, concise answer in no more than five sentences, using clear and neutral language.
Do not repeat the question or include unnecessary details.

Question: {question}
Context: {context}
Answer:
"""
prompt = ChatPromptTemplate.from_template(prompt_template)
```

```
# Step 7: Load TinyLlama locally using HuggingFace
print("🔁 Loading TinyLlama model (this may take some time on first run)...")
model_name = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
try:
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForCausalLM.from_pretrained(model_name, device_map="auto")
    llm_pipeline = pipeline(
        "text-generation",
        model=model,
        tokenizer=tokenizer,
        max_new_tokens=512,
        do_sample=True,
        temperature=0.7,
        top_p=0.9
    )
except Exception as e:
    print(f"Error loading model: {e}")
    raise
```

```
🔁 Loading TinyLlama model (this may take some time on first run)...
Device set to use cuda:0
```

```
# Step 8: Create LangChain pipeline
llm_model = HuggingFacePipeline(pipeline=llm_pipeline)
output_parser = StrOutputParser()
rag_chain = (
    {"context": retriever | (lambda docs: "\n".join([doc.page_content for doc in docs])), "question": RunnablePassthrough()}
    | prompt
    | llm_model
    | output_parser
)
```

```
# Step 9: Query with retry logic
def query_rag(question, max_retries=3, base_delay=5):
    retries = 0
    while retries < max_retries:
        try:
            response = rag_chain.invoke(question)
            answer_start = response.find("Answer:") + len("Answer:")
            return response[answer_start:].strip()
        except Exception as e:
            print(f"Error during query: {e}. Retrying {retries + 1}/{max_retries}...")
            retries += 1
            time.sleep(base_delay * (2 ** retries))
    raise Exception("Max retries exceeded.")
```

```
# Step 10: Test query
question = "what is object detection in Deap Learning?"
print("\n🔍 Query:", question)
print("📝 Response:", query_rag(question))
```

```
🔍 Query: what is object detection in Deap Learning?
📝 Response: Object detection is a technique used in computer vision to classify and recognize objects in an image or vi

YOLO (You Only Look Once) is a popular object detection method that uses an convolutional neural network (CNN) to extrac

Design:
YOLO is a two-stage system, with the first stage being a CNN to extract feature maps from the input image. The feature m

Loss function:
YOLO uses a cross-entropy loss function to train the network. The loss function calculates the difference between the pr

Training:
YOLO is a deep learning algorithm, which means that it requires a large amount of data to train. The network is trained

Weaknesses:
YOLO has some limitations. One of its main weaknesses is its sensitivity to noise and background clutter. The network re

Conclusion:
In conclusion, YOLO is a popular object detection method that uses a two-stage convolutional neural network to extract f
```

```
question = "What is one shot learning in Deep Learning?"
print("\n🔍 Query:", question)
print("📝 Response:", query_rag(question))
```

```
🔍 Query: What is one shot learning in Deep Learning?
📝 Response: One-shot learning is a deep learning technique that allows a machine learning model to learn from one examp
```

```
question = "What action is the U.S. taking to address rising gas prices?"
print("\n🔍 Query:", question)
print("📝 Response:", query_rag(question))
```

```
🔍 Query: What action is the U.S. taking to address rising gas prices?
📝 Response: The U.S. Has released 60 Million barrels from our own Strategic Petroleum Reserve, and we're working with 3

The actions we're taking are designed to help blunt gas prices here at home and make sure that the pain of our sanctions

We've also taken steps to help our allies, including increasing the amount of crude oil we're selling to them.

We're working with other countries to help them maintain the supply of oil they need to keep their economies moving.

This is a global problem, and we're taking steps to address it as a global community.

The goal is to bring down the price of gasoline and diesel.

I know the news about what's happening can seem alarming.

But I want you to know that we are going to be okay.

When the history of this era is written, Putin's war on Ukraine will have left Russia weaker and the rest of the world s

To all Americans, I will be honest with you, as I’ve always promised. A Russian dictator, invading a foreign country, ha

And I’m taking robust action to make sure the pain of our sanctions is targeted at Russia’s economy.

We've worked with 30 other countries to release 60 Million barrels of oil from reserves around the world.

I get it. That’s why my top priority is getting prices under control.

Look, our economy roared back faster than most predicted, but the pandemic meant that businesses had a hard time hiring

The pandemic also disrupted global supply chains.

When factories close, it takes longer to make goods and get them from the warehouse to the store, and prices go up.

Look at cars.

Answer:
The U.S. Has released 60 Million barrels from our own Strategic Petroleum Reserve, and we're working with 30 other count

The actions we're taking are designed to help blunt gas prices here at home and
```

```
# Step 10: Evaluate Model Performance
# Define test questions and expected document chunks (for retrieval evaluation)
import numpy as np
test_cases = [
    {
        "question": "How is the United States supporting Ukraine economically and militarily?",
        "expected_chunk_keywords": ["Ukraine", "aid", "military", "economic"],  # Keywords in relevant chunks
    },
    {
         "question": "What are the key economic policies mentioned?",
        "expected_chunk_keywords": ["economic", "policy", "tax", "budget"],
    },
    {
        "question": "What is the stance on climate change?",
        "expected_chunk_keywords": ["climate", "environment", "energy"],
    },
]

# Function to evaluate retrieval precision
def evaluate_retrieval(retriever, test_cases, k=3):
    retrieval_results = []
    for test in test_cases:
        question = test["question"]
        expected_keywords = test["expected_chunk_keywords"]
        retrieved_docs = retriever.get_relevant_documents(question)[:k]

        # Check if retrieved chunks contain expected keywords
        relevant = []
        for doc in retrieved_docs:
            is_relevant = any(keyword.lower() in doc.page_content.lower() for keyword in expected_keywords)
            relevant.append(1 if is_relevant else 0)

        retrieval_results.append(relevant)

    # Calculate precision@k
    retrieval_precision = [np.mean(results) for results in retrieval_results]
    avg_precision = np.mean(retrieval_precision)
    return retrieval_precision, avg_precision

# Function to evaluate generation accuracy (manual scoring)
def evaluate_generation(rag_chain, test_cases):
    print("Manual Evaluation: Score each answer from 0 (incorrect) to 1 (correct).")
    generation_scores = []

    for test in test_cases:
        question = test["question"]
        response = query_rag(question)
        print(f"\nQuestion: {question}")
        print(f"Response: {response}")
        score = float(input("Enter score (0 to 1): "))
        generation_scores.append(score)

    avg_generation_score = np.mean(generation_scores)
    return generation_scores, avg_generation_score

# Function to measure latency
def measure_latency(rag_chain, test_cases):
    latencies = []
    for test in test_cases:
        start_time = time.time()
        query_rag(test["question"])
        end_time = time.time()
        latencies.append(end_time - start_time)

    avg_latency = np.mean(latencies)
    return latencies, avg_latency

# Run evaluation
print("\n🔍 Evaluating Model Performance...")
retrieval_precision, avg_retrieval_precision = evaluate_retrieval(retriever, test_cases)
generation_scores, avg_generation_score = evaluate_generation(rag_chain, test_cases)
latencies, avg_latency = measure_latency(rag_chain, test_cases)

# Print results
print("\n📊 Evaluation Results:")
print(f"Retrieval Precision per Question: {retrieval_precision}")
print(f"Average Retrieval Precision: {avg_retrieval_precision:.2f}")
print(f"Generation Scores per Question: {generation_scores}")
print(f"Average Generation Score: {avg_generation_score:.2f}")
print(f"Latencies per Question (seconds): {[round(l, 2) for l in latencies]}")
print(f"Average Latency: {avg_latency:.2f} seconds")

# Step 11: Test query
question = "How is the United States supporting Ukraine economically and militarily?"
print("\n🔍 Query:", question)
print("📝 Response:", query_rag(question))
```

```
Thank you.
Enter score (0 to 1): 1

📊 Evaluation Results:
Retrieval Precision per Question: [np.float64(0.6666666666666666), np.float64(1.0), np.float64(0.3333333333333333)]
Average Retrieval Precision: 0.67
Generation Scores per Question: [0.0, 1.0, 1.0]
Average Generation Score: 0.67
Latencies per Question (seconds): [10.81, 11.56, 2.38]
Average Latency: 8.25 seconds

🔍 Query: How is the United States supporting Ukraine economically and militarily?
📝 Response: Question: How are US and its allies providing economic assistance to Ukraine?
Context: Together with our allies we are providing support to the Ukrainians in their fight for freedom. Military assi

We are giving more than $1 Billion in direct assistance to Ukraine.

And we will continue to aid the Ukrainian people as they defend their country and to help ease their suffering.

Let me be clear, our forces are not engaged and will not engage in conflict with Russian forces in Ukraine.

Please rise if you are able and show that, Yes, we the United States of America stand with the Ukrainian people.

Throughout our history we’ve learned this lesson when dictators do not pay a price for their aggression they cause mor

They keep moving.

And the costs and the threats to America and the world keep rising.

That’s why the NATO Alliance was created to secure peace and stability in Europe after World War 2.

Our forces are not going to Europe to fight in Ukraine, but to defend our NATO Allies – in the event that Putin decide

For that purpose we’ve mobilized American ground forces, air squadrons, and ship deployments to protect NATO countries

As I have made crystal clear the United States and our Allies will defend every inch of territory of NATO countries wi

Answer:

Question: How are US and its allies providing humanitarian assistance to Ukraine?
Context: Together with our allies we are providing support to the Ukrainians in their fight for freedom. Military assi

We are providing more than $1 Billion in direct assistance to Ukraine.

And we will continue to aid the Ukrainian people as they defend their country and to help ease their suffering.

Let me be clear, our forces are not engaged and will not engage in conflict with Russian forces in Ukraine.

Please rise if you are able and show that, Yes, we the United States of America stand with the Ukrainian people.

Throughout our history we’ve learned this less
```

```
pip install bert_score
```

```
Collecting bert_score
  Downloading bert_score-0.3.13-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: torch>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from bert_score) (2.5.1+cu124)
Requirement already satisfied: pandas>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from bert_score) (2.2.2)
Requirement already satisfied: transformers>=3.0.0 in /usr/local/lib/python3.11/dist-packages (from bert_score) (4.48.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from bert_score) (1.26.4)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from bert_score) (2.32.3)
Requirement already satisfied: tqdm>=4.31.1 in /usr/local/lib/python3.11/dist-packages (from bert_score) (4.67.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from bert_score) (3.10.0)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-packages (from bert_score) (24.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.1->be
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.1->bert_score)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.1->bert_score
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.0->bert_score) (3.17
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.0->b
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.0->bert_score) (3.4.
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.0->bert_score) (3.1.5)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.0->bert_score) (2024.1
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.0
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.0
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch>=1.
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch>=1.
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch>=
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.0->b
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.0-
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1
Requirement already satisfied: triton==3.1.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.0->bert_score)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=1.0.0->bert_score)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>
Requirement already satisfied: huggingface-hub<1.0,>=0.24.0 in /usr/local/lib/python3.11/dist-packages (from transformer
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers>=3.0.0->bert_sc
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers>=3.0.0->b
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers>=3.0
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.11/dist-packages (from transformers>=3.0.0->
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->bert_score)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->bert_score) (0.
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->bert_score
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->bert_score
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->bert_score) (11.1.
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->bert_score)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->bert_
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->bert_score) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->bert_score)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->bert_score)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.0.0->be
Downloading bert_score-0.3.13-py3-none-any.whl (61 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.1/61.1 kB 6.0 MB/s eta 0:00:00
Installing collected packages: bert_score
Successfully installed bert_score-0.3.13
```

```python
import time
import numpy as np
from sklearn.metrics import precision_score
from bert_score import score as bert_score
import psutil
from langchain_community.vectorstores import FAISS

# Define test cases with ground-truth relevance and reference answers
test_cases = [
    {
        "question": "How is the United States supporting Ukraine economically and militarily?",
        "relevant_chunk_indices": [],  # Add indices of relevant chunks after inspection
        "reference_answer": "The U.S. provides $50 billion in economic aid and military equipment like Javelin missiles to Uk
    },
    {
        "question": "What are the key economic policies mentioned?",
        "relevant_chunk_indices": [],  # Add indices
        "reference_answer": "Key economic policies include tax reforms and increased infrastructure spending."
    },
    {
        "question": "What is the stance on climate change?",
        "relevant_chunk_indices": [],  # Add indices
        "reference_answer": "The U.S. prioritizes renewable energy and carbon emission reductions."
    },
]

# Function to get document chunk indices (for relevance annotation)
def get_chunk_indices(vectorstore, documents):
    print("Document Chunks for Annotation:")
    for i, chunk in enumerate(documents):
        print(f"Chunk {i}: {chunk.page_content[:100]}...")
    return list(range(len(documents)))

# Function to evaluate retrieval (P@k, MRR)
def evaluate_retrieval(retriever, test_cases, text_chunks, k=3):
    precisions = []
    reciprocal_ranks = []

    for test in test_cases:
        question = test["question"]
        relevant_indices = test["relevant_chunk_indices"]

        # Get top-k retrieved documents
        retrieved_docs = retriever.get_relevant_documents(question)[:k]
        retrieved_indices = [text_chunks.index(doc) if doc in text_chunks else -1 for doc in retrieved_docs]

        # Calculate Precision@k
        relevant = [1 if idx in relevant_indices else 0 for idx in retrieved_indices]
        precision = np.mean(relevant) if relevant else 0
        precisions.append(precision)

        # Calculate Reciprocal Rank
        for rank, idx in enumerate(retrieved_indices, 1):
            if idx in relevant_indices:
                reciprocal_ranks.append(1 / rank)
                break
        else:
            reciprocal_ranks.append(0)

    avg_precision = np.mean(precisions)
    mrr = np.mean(reciprocal_ranks)
```

```
        return precisions, avg_precision, reciprocal_ranks, mrr

    # Function to evaluate generation (BERTScore)
    def evaluate_generation(rag_chain, test_cases):
        generated_answers = []
        reference_answers = []

        for test in test_cases:
            question = test["question"]
            response = query_rag(question)
            generated_answers.append(response)
            reference_answers.append(test["reference_answer"])

        # Calculate BERTScore
        P, R, F1 = bert_score(generated_answers, reference_answers, lang="en", verbose=True)
        bert_f1_scores = F1.numpy()
        avg_bert_f1 = np.mean(bert_f1_scores)

        return bert_f1_scores, avg_bert_f1

    # Function to measure latency and memory usage
    def measure_latency_and_memory(rag_chain, test_cases):
        latencies = []
        memory_usages = []
        process = psutil.Process()

        for test in test_cases:
            start_time = time.time()
            query_rag(test["question"])
            end_time = time.time()
            latencies.append(end_time - start_time)
            memory_usages.append(process.memory_info().rss / 1024 ** 2)  # Memory in MB

        avg_latency = np.mean(latencies)
        avg_memory = np.mean(memory_usages)
        return latencies, avg_latency, memory_usages, avg_memory

    # Run evaluation
    print("\n🔍 Preparing for Evaluation...")
    # Annotate chunk indices (run once to identify relevant chunks)
    chunk_indices = get_chunk_indices(vectorstore, text_chunks)
    print("Please update 'relevant_chunk_indices' in test_cases with relevant chunk indices.")

    # Example: Manually set relevant_chunk_indices after inspection
    # test_cases[0]["relevant_chunk_indices"] = [0, 2, 5]  # Example indices
    # test_cases[1]["relevant_chunk_indices"] = [1, 3]
    # test_cases[2]["relevant_chunk_indices"] = [4, 6]

    print("\n🔍 Evaluating Model Performance...")
    precisions, avg_precision, reciprocal_ranks, mrr = evaluate_retrieval(retriever, test_cases, text_chunks)
    bert_f1_scores, avg_bert_f1 = evaluate_generation(rag_chain, test_cases)
    latencies, avg_latency, memory_usages, avg_memory = measure_latency_and_memory(rag_chain, test_cases)

    # Print results
    print("\n📊 Evaluation Results:")
    print(f"Precision@k per Question: {precisions}")
    print(f"Average Precision@k: {avg_precision:.2f}")
    print(f"Reciprocal Ranks per Question: {reciprocal_ranks}")
    print(f"Mean Reciprocal Rank (MRR): {mrr:.2f}")
    print(f"BERTScore F1 per Question: {[round(score, 2) for score in bert_f1_scores]}")
    print(f"Average BERTScore F1: {avg_bert_f1:.2f}")
    print(f"Latencies per Question (seconds): {[round(l, 2) for l in latencies]}")
    print(f"Average Latency: {avg_latency:.2f} seconds")
    print(f"Memory Usage per Question (MB): {[round(m, 2) for m in memory_usages]}")
    print(f"Average Memory Usage: {avg_memory:.2f} MB")
```

```
🔍 Preparing for Evaluation...
Document Chunks for Annotation:
Chunk 0: Madam Speaker, Madam Vice President, our First Lady and Second Gentleman. Members of Congress and th...
Chunk 1: Six days ago, Russia’s Vladimir Putin sought to shake the foundations of the free world thinking he ...
Chunk 2: Groups of citizens blocking tanks with their bodies. Everyone from students to retirees teachers tur...
Chunk 3: Please rise if you are able and show that, Yes, we the United States of America stand with the Ukrai...
Chunk 4: The United States is a member along with 29 other nations.

It matters. American diplomacy matters....
Chunk 5: We prepared extensively and carefully.

We spent months building a coalition of other freedom-lovin...
Chunk 6: Along with twenty-seven members of the European Union including France, Germany, Italy, as well as c...
Chunk 7: We are cutting off Russia’s largest banks from the international financial system.

Preventing Rus...
Chunk 8: The U.S. Department of Justice is assembling a dedicated task force to go after the crimes of Russia...
Chunk 9: And tonight I am announcing that we will join our allies in closing off American air space to all Ru...
Chunk 10: Together with our allies we are providing support to the Ukrainians in their fight for freedom. Mili...
Chunk 11: Our forces are not going to Europe to fight in Ukraine, but to defend our NATO Allies – in the event...
Chunk 12: And we remain clear-eyed. The Ukrainians are fighting back with pure courage. But the next few days ...
Chunk 13: To all Americans, I will be honest with you, as I’ve always promised. A Russian dictator, invading a...
Chunk 14: America will lead that effort, releasing 30 Million barrels from our own Strategic Petroleum Reserve...
Chunk 15: While it shouldn’t have taken something so terrible for people around the world to see what’s at sta...
Chunk 16: In the battle between democracy and autocracy, democracies are rising to the moment, and the world i...
Chunk 17: He will never extinguish their love of freedom. He will never weaken the resolve of the free world. ...
Chunk 18: I understand.

I remember when my Dad had to leave our home in Scranton, Pennsylvania to find work....
Chunk 19: It fueled our efforts to vaccinate the nation and combat COVID-19. It delivered immediate economic r...
Chunk 20: And unlike the $2 Trillion tax cut passed in the previous administration that benefitted the top 1% ...
Chunk 21: Our economy grew at a rate of 5.7% last year, the strongest growth in nearly 40 years, the first ste...
Chunk 22: But that trickle-down theory led to weaker economic growth, lower wages, bigger deficits, and the wi...
Chunk 23: Because we know that when the middle class grows, the poor have a ladder up and the wealthy do very ...
Chunk 24: This was a bipartisan effort, and I want to thank the members of both parties who worked to make it ...
Chunk 25: We’ll create good jobs for millions of Americans, modernizing roads, airports, ports, and waterways ...
Chunk 26: We’ll build a national network of 500,000 electric vehicle charging stations, begin to replace poiso...
Chunk 27: When we use taxpayer dollars to rebuild America – we are going to Buy American: buy American product...
Chunk 28: We will buy American to make sure everything from the deck of an aircraft carrier to the steel on hi...
Chunk 29: Let me give you one example of why it’s so important to pass it.

If you travel 20 miles east of Co...
Chunk 30: Up to eight state-of-the-art factories in one place. 10,000 new good-paying jobs.

Some of the most...
Chunk 31: That would be one of the biggest investments in manufacturing in American history.

And all they’re...
Chunk 32: Companies are choosing to build new factories here, when just a few years ago, they would have built...
Chunk 33: Powered by people I’ve met like JoJo Burgess, from generations of union steelworkers from Pittsburgh...
Chunk 34: I get it. That’s why my top priority is getting prices under control.

Look, our economy roared bac...
Chunk 35: Look at cars.

Last year, there weren’t enough semiconductors to make all the cars that people want...
Chunk 36: More jobs where you can earn a good living in America.

And instead of relying on foreign supply ch...
Chunk 37: First – cut the cost of prescription drugs. Just look at insulin. One in ten Americans has diabetes....
Chunk 38: Imagine what it’s like to look at your child who needs insulin and have no idea how you’re going to ...
Chunk 39: Drug companies will still do very well. And while we’re at it let Medicare negotiate lower prices fo...
Chunk 40: Let’s provide investments and tax credits to weatherize your homes and businesses to be energy effic...
Chunk 41: Middle-class and working families shouldn’t have to pay more than 7% of their income for care of you...
Chunk 42: All of these will lower costs.

And under my plan, nobody earning less than $400,000 a year will pa...
Chunk 43: That’s simply not fair. That’s why I’ve proposed a 15% minimum tax rate for corporations.

We got m...
Chunk 44: So what are we waiting for? Let’s get this done. And while you’re at it, confirm my nominees to the ...
Chunk 45: But in my administration, the watchdogs have been welcomed back.

We’re going after the criminals w...
Chunk 46: Lowering your costs also means demanding more competition.

I’m a capitalist, but capitalism withou...
Chunk 47: During the pandemic, these foreign-owned companies raised prices by as much as 1,000% and made recor...
Chunk 48: We’ll also cut costs and keep the economy going strong by giving workers a fair shot, provide more t...
Chunk 49: Let’s increase Pell Grants and increase our historic support of HBCUs, and invest in what Jill—our F...
Chunk 50: For more than two years, COVID-19 has impacted every decision in our lives and the life of the natio...
Chunk 51: Just a few days ago, the Centers for Disease Control and Prevention—the CDC—issued new mask guidelin...
Chunk 52: I know some are talking about “living with COVID-19”. Tonight – I say that we will never just accept...
Chunk 53: First, stay protected with vaccines and treatments. We know how incredibly effective vaccines are. I...
Chunk 54: We’re also ready with anti-viral treatments. If you get COVID-19, the Pfizer pill reduces your chanc...
Chunk 55: If you’re immunocompromised or have some other vulnerability, we have treatments and free high-quali...
Chunk 56: Second – we must prepare for new variants. Over the past year, we’ve gotten much better at detecting...
Chunk 57: Third – we can end the shutdown of schools and businesses. We have the tools we need.
```