Data and Artificial Intelligence

Cyber Shujaa Program

Assignment 11: Natural Language Processing

Student Name: Kelvin Shilisia

Student ID: cs-da02 -25066

Introduction

In this assignment, I will apply my understanding of Generative AI and Retrieval-Augmented Generation (RAG) to build a practical pipeline that retrieves relevant document chunks and generates context-aware answers:

By completing this assignment, I will demonstrate my ability to:

Apply generative AI concepts to synthesize answers from retrieved document content.

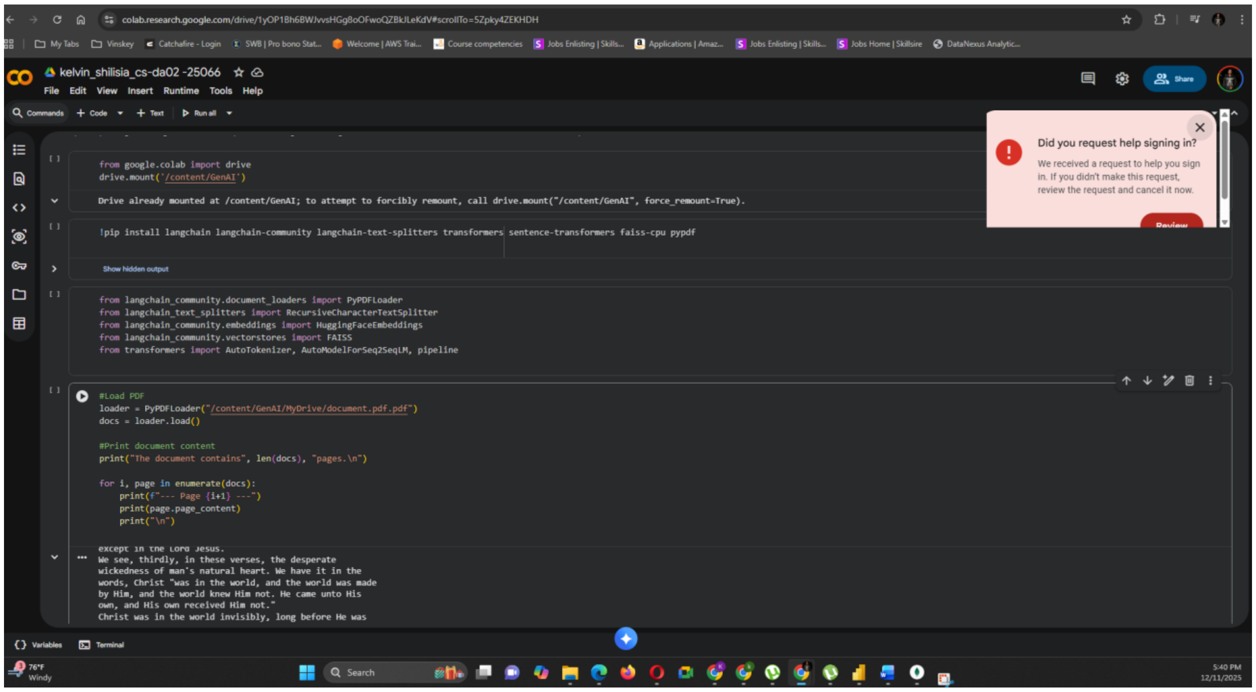
Extract key information from a PDF document by splitting it into chunks, embedding it using Sentence-Transformers, and storing it in a FAISS vector database for efficient retrieval.

Demonstrate how retrieval improves generative question-answering by comparing answers generated from document-grounded context versus generic answers.

Implement a complete RAG pipeline in Python using LangChain, Hugging Face Transformers, and FAISS.

Practice prompt engineering to structure queries that guide the generative model for clearer and more detailed responses.

Importing libraries



```
from google.colab import drive
drive.mount('/content/GenAI')
```

```
Drive already mounted at /content/GenAI; to attempt to forcibly remount, call drive.mount("/content/GenAI", force_remount=True).
```

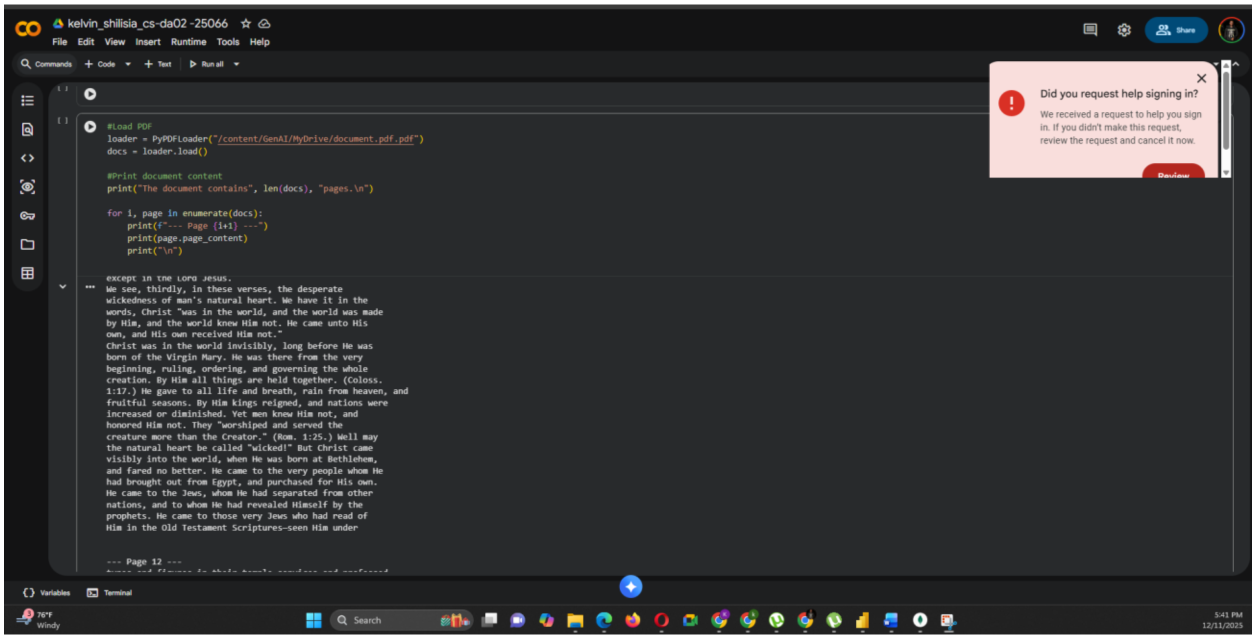
```
!pip install langchain langchain-community langchain-text-splitters transformers sentence-transformers faiss-cpu pypdf
```

```
from langchain_community.document_loaders import PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM, pipeline
```

```
#Load PDF
loader = PyPDFLoader("/content/GenAI/MyDrive/document.pdf.pdf")
docs = loader.load()

#Print document content
print("The document contains", len(docs), "pages.\n")

for i, page in enumerate(docs):
    print(f"--- Page {i+1} ---")
    print(page.page_content)
    print("\n")
```

```
except in the Lord Jesus.
We see, thirdly, in these verses, the desperate
wickedness of man's natural heart. We have it in the
words, Christ "was in the world, and the world was made
by Him, and the world knew Him not. He came unto His
own, and His own received Him not."
Christ was in the world invisibly, long before He was
```

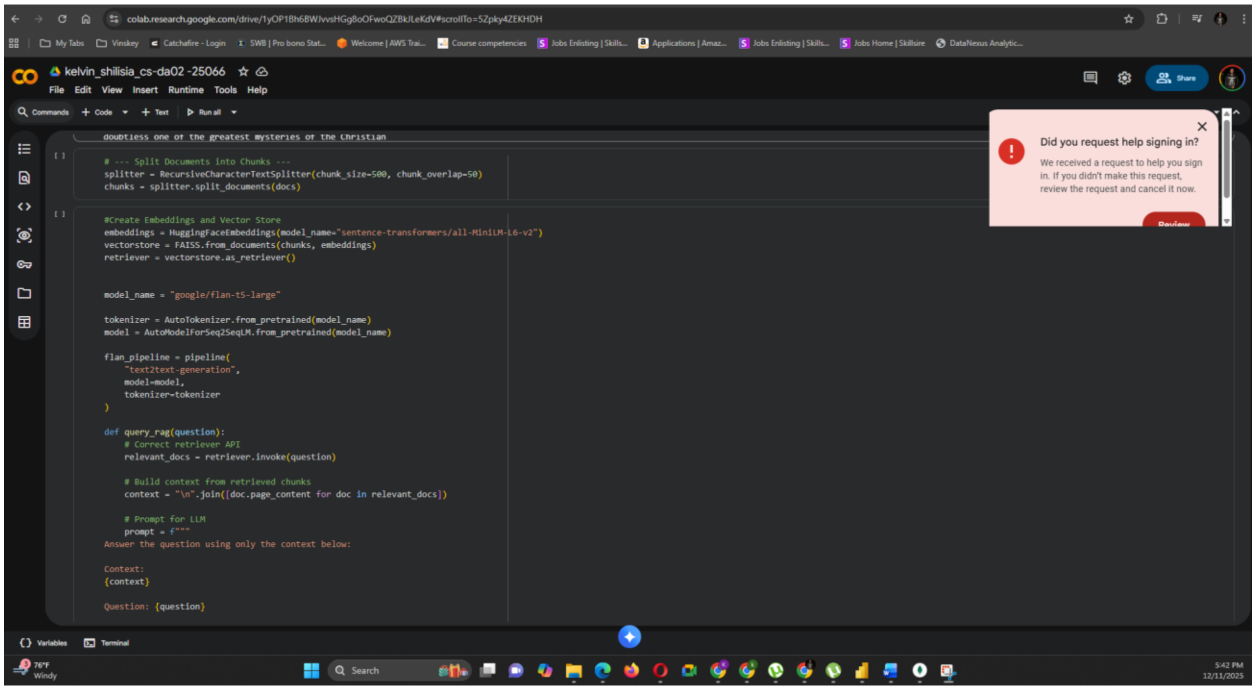
Loading the PDF file



```
#Load PDF
loader = PyPDFLoader("/content/GenAI/MyDrive/document.pdf.pdf")
docs = loader.load()

#Print document content
print("The document contains", len(docs), "pages.\n")

for i, page in enumerate(docs):
    print(f"--- Page {i+1} ---")
    print(page.page_content)
    print("\n")
```

```
except in the Lord Jesus.
We see, thirdly, in these verses, the desperate
wickedness of man's natural heart. We have it in the
words, Christ "was in the world, and the world was made
by Him, and the world knew Him not. He came unto His
own, and His own received Him not."
Christ was in the world invisibly, long before He was
born of the Virgin Mary. He was there from the very
beginning, ruling, ordering, and governing the whole
creation. By Him all things are held together. (Coloss.
1:17.) He gave to all life and breath, rain from heaven, and
fruitful seasons. By Him kings reigned, and nations were
increased or diminished. Yet men knew Him not, and
honored Him not. They "worshiped and served the
creature more than the Creator." (Rom. 1:25.) Well may
the natural heart be called "wicked!" But Christ came
visibly into the world, when He was born at Bethlehem,
and fared no better. He came to the very people whom He
had brought out from Egypt, and purchased for His own.
He came to the Jews, whom He had separated from other
nations, and to whom He had revealed Himself by the
prophets. He came to those very Jews who had read of
Him in the Old Testament Scriptures—seen Him under


--- Page 12 ---
```

Splitting the document, creating embeddings and Vector Store


kelvin_shilisia_cs-da02 -25066
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Variables Terminal

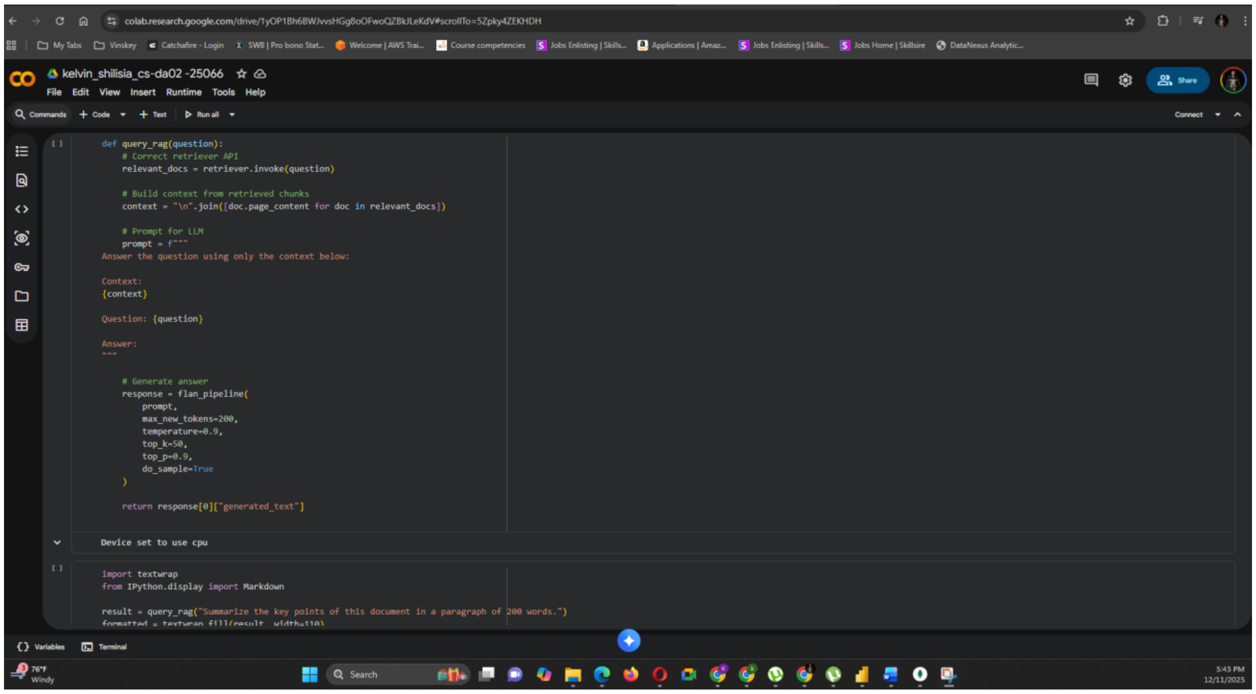
```
doubtless one of the greatest mysteries of the Christian
```

```python
# --- Split Documents into Chunks ---
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
chunks = splitter.split_documents(docs)
```

```python
#Create Embeddings and Vector Store
embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
vectorstore = FAISS.from_documents(chunks, embeddings)
retriever = vectorstore.as_retriever()


model_name = "google/flan-t5-large"

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

flan_pipeline = pipeline(
    "text2text-generation",
    model=model,
    tokenizer=tokenizer
)

def query_rag(question):
    # Correct retriever API
    relevant_docs = retriever.invoke(question)

    # Build context from retrieved chunks
    context = "\n".join([doc.page_content for doc in relevant_docs])

    # Prompt for LLM
    prompt = f"""
Answer the question using only the context below:

Context:
{context}

Question: {question}
```

Did you request help signing in?

We received a request to help you sign in. If you didn't make this request, review the request and cancel it now.

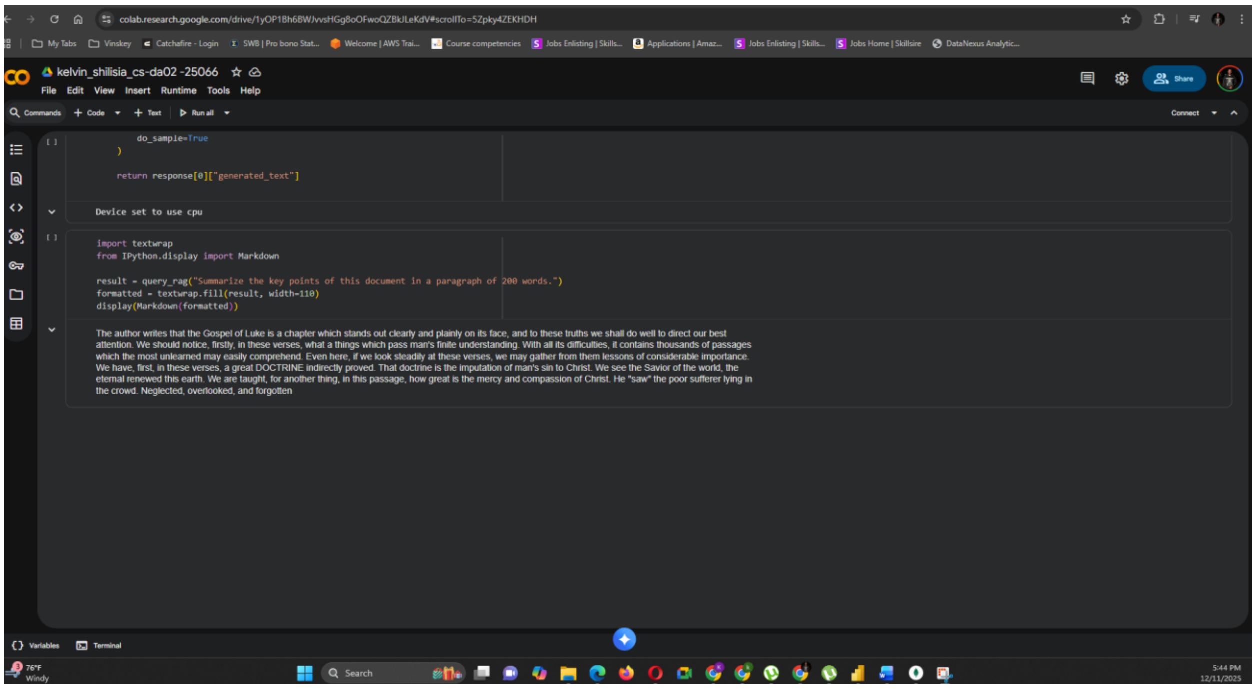

kelvin_shilisia_cs-da02 -25066
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Connect
Variables Terminal

```python
def query_rag(question):
    # Correct retriever API
    relevant_docs = retriever.invoke(question)

    # Build context from retrieved chunks
    context = "\n".join([doc.page_content for doc in relevant_docs])

    # Prompt for LLM
    prompt = f"""
Answer the question using only the context below:

Context:
{context}

Question: {question}

Answer:
"""

    # Generate answer
    response = flan_pipeline(
        prompt,
        max_new_tokens=200,
        temperature=0.9,
        top_k=50,
        top_p=0.9,
        do_sample=True
    )

    return response[0]["generated_text"]
```

```
Device set to use cpu
```

```python
import textwrap
from IPython.display import Markdown

result = query_rag("Summarize the key points of this document in a paragraph of 200 words.")
formatted = textwrap.fill(result, width=110)
```

Summarizing the document


kelvin_shilisia_cs-da02 -25066
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```
        do_sample=True
    )

    return response[0]["generated_text"]
```

```
Device set to use cpu
```

```
import textwrap
from IPython.display import Markdown

result = query_rag("Summarize the key points of this document in a paragraph of 200 words.")
formatted = textwrap.fill(result, width=110)
display(Markdown(formatted))
```

The author writes that the Gospel of Luke is a chapter which stands out clearly and plainly on its face, and to these truths we shall do well to direct our best attention. We should notice, firstly, in these verses, what a things which pass man's finite understanding. With all its difficulties, it contains thousands of passages which the most unlearned may easily comprehend. Even here, if we look steadily at these verses, we may gather from them lessons of considerable importance. We have, first, in these verses, a great DOCTRINE indirectly proved. That doctrine is the imputation of man's sin to Christ. We see the Savior of the world, the eternal renewed this earth. We are taught, for another thing, in this passage, how great is the mercy and compassion of Christ. He "saw" the poor sufferer lying in the crowd. Neglected, overlooked, and forgotten

Link: https://colab.research.google.com/drive/1yOP1Bh6BWJvvsHGg8oOFwoQZBkJLeKdV?usp=sharing

Summary Report

Data Analysis Key Findings

- **Project Scope and Objectives**: The project aims to build a practical Retrieval-Augmented Generation (RAG) pipeline. Key objectives include applying generative AI concepts to synthesize answers, extracting information from PDFs (splitting, embedding using Sentence-Transformers, storing in FAISS), demonstrating the benefits of retrieval in generative QA, implementing the RAG pipeline using LangChain, Hugging Face Transformers, and FAISS, and practicing prompt engineering.
- **Implemented RAG Pipeline Structure**: The RAG pipeline involves several stages:
    - **Environment Setup**: Google Drive was successfully mounted, and necessary libraries like langchain, transformers, sentence-transformers, faiss-cpu, and pypdf were installed.
    - **Document Loading**: A PDF document named "document.pdf.pdf" containing 401 pages was successfully loaded using PyPDFLoader.

    - **Document Processing (Defined)**: Code for chunking documents using RecursiveCharacterTextSplitter with chunk_size=500 and chunk_overlap=50 is defined.
    - **Embedding and Vector Store (Defined)**: Code to create embeddings with HuggingFaceEmbeddings using the "sentence-transformers/all-MiniLM-L6-v2" model and to build a FAISS vector store with a retriever is defined.
    - **Generative Model Setup (Defined)**: The google/flan-t5-large model and tokenizer are configured, and a text2text-generation pipeline, along with a query_rag function (with parameters like max_new_tokens=200, temperature=0.9), are defined.
- **Current Progress Status**:
    - **Executed Steps**: Google Drive mounting, library installation, and PDF document loading (of a 401-page PDF) have been successfully completed.
    - **Defined but Unexecuted Steps**: The steps for document chunking, creating embeddings and the FAISS vector store, setting up the Flan-T5-Large text generation model, defining the query_rag function, and executing the initial RAG query for summarization have been defined in the notebook cells but have not yet been run.

Insights or Next Steps

- **Execution of Core RAG Components**: The immediate next step is to execute the defined cells for document chunking, embedding generation, FAISS vector store creation, and the generative model setup to fully operationalize the RAG pipeline.
- **Performance Evaluation and Enhancement**: Future work should focus on evaluating the RAG model's performance using retrieval and generation metrics, expanding the document base beyond PDFs, refining chunking strategies (e.g., context-aware chunking), exploring alternative embedding and generative models, and enhancing prompt engineering techniques.