Technical Report on Multi-Format Document Retrieval-Augmented Generation Pipeline

Introduction

This project presents a Retrieval-Augmented Generation (RAG) pipeline designed to handle documents in multiple formats — PDF, DOCX, and PPTX. The pipeline ingests these files directly from Google Drive, processes them into semantic units, encodes them as embeddings, and stores them in a vector database. User queries are then answered by retrieving the most relevant content and grounding a language model's output in that context.

Data Ingestion

Since the pipeline is designed for Google Colab, integration with Google Drive allows convenient access to files. The Drive is mounted, and a directory path is set for the project's document repository. This ensures that all source materials remain accessible across sessions.

Code Block:

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

# 👇 change only if your Drive path differs
DATA_DIR = "/content/drive/MyDrive/project_two"
```

Document Loading

Dedicated loaders are used to process each document format:

- **PyPDFLoader** extracts structured text from PDF files.
- **Docx2txtLoader** converts Microsoft Word documents into raw text.
- **python-pptx** reads PowerPoint slides and retrieves textual content.

This multi-loader approach allows the system to build a uniform text corpus from heterogeneous sources.

Code Block:

```
from pathlib import Path
from typing import List
import re

from langchain_core.documents import Document
from langchain_community.document_loaders import PyPDFLoader,
Docx2txtLoader
from pptx import Presentation
```

Text Processing and Splitting

Once text is extracted, it undergoes preprocessing to remove unnecessary whitespace, control characters, and artifacts. A recursive character text splitter then segments the cleaned text into chunks of ~1,000 characters with 150-character overlaps. This strategy ensures semantic coherence while staying within the input limits of transformer-based models.

Code Block:

```
from langchain_text_splitters import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=150,
    separators=["\n\n", "\n", " ", ""]
)
chunks = splitter.split_documents(docs)
```

Embedding Generation

Each chunk is transformed into a dense semantic vector using HuggingFace's all-MiniLM-L6-v2 model. GPU acceleration in Colab significantly reduces embedding time for larger document collections.

Code Block:

```
from langchain_community.embeddings import HuggingFaceEmbeddings

emb = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2",
    model_kwargs={"device": "cuda"}  # fallback "cpu"
)
```

Vector Database (ChromaDB)

The embeddings, together with metadata (e.g., file name, source path), are stored in ChromaDB. This persistent vector database allows efficient semantic similarity search, enabling the retriever to identify the most relevant context for a given query.

Code Block:

```
from langchain_community.vectorstores import Chroma

vectorstore = Chroma.from_documents(
    documents=chunks,
    embedding=emb,
    collection_name="project_two_docs",
    persist_directory="chroma_db"
)
retriever = vectorstore.as_retriever(search_kwargs={"k": 5})
```

Retrieval and Question Answering

When a user poses a question, the retriever selects the top-k most relevant document chunks. These are combined into a structured prompt that includes both the retrieved context and the user query. The language model is instructed to provide an answer strictly based on the retrieved content. Alongside the answer, the original source documents are cited for transparency.

Code Block:

```
def rag_answer(question, k=5, max_chars=1000):
    docs = retriever.get_relevant_documents(question)
    context = "\n\n".join(
        f"[{i+1}]
{d.metadata.get('source','')}\n{d.page_content[:max_chars]}"
        for i, d in enumerate(docs)
    )
    prompt = (
        "You are a helpful assistant. Answer using ONLY the context.\n\n"
        f"Context:\n{context}\n\nQuestion: {question}\n\nAnswer:"
    )
    # Replace ai.generate_text(prompt) with your chosen LLM
    reply = "⚠️ Placeholder: integrate with OpenAI or Colab AI"
    return reply, docs
```

Applications

- **Education**: Rapid question answering across lecture slides, assignments, and study notes.
- **Enterprise**: Intelligent assistants for policy documents, training manuals, and reports.

- **Research**: Semantic exploration of collections of academic papers and presentations.

Conclusion

This project demonstrates how diverse document formats can be unified within a single RAG framework. It illustrates expertise in file ingestion, natural language preprocessing, embedding models, vector storage, and retrieval-based LLM prompting. By delivering grounded, source-referenced answers, the pipeline offers both accuracy and trustworthiness.

In [ ]: