# **Retrieval Augmented Generation (RAG) Challenge**

Collaborators: Natalia Torres, Paola Rivera

**Project Overview**: The purpose of this project is to implement a Retrieval-Augmented Generation (RAG) system using a dataset containing European financial laws. The system enhances Large Language Model (LLM) responses by retrieving relevant legal text passages and providing context-aware answers.

**Objectives:**

* Extract text from financial law PDFs
* Preprocess and structure the data for efficient retrieval
* Create embeddings and store them in a vector database
* Implement retrieval mechanisms to fetch relevant text
* Integrate with an LLM to enhance response generation

## **Data Selection and Exploration**

The dataset contains **legal and regulatory texts** related to:

* **AML (Anti-Money Laundering)**
* **CRR/CRD (Capital Requirements for Banks & Investment Firms)**
* **PSD (Payment Services Directive)**
* **MCD (Mortgage Credit Directive)**
* **IFR/IFD (Investment Firm Regulations)**
* **Deposit Guarantee Schemes**
* **Securitization Regulations**

These texts are **highly structured**, containing:

* **Definitions & Legal Clauses** (e.g., definitions of financial instruments)
* **Compliance Requirements** (e.g., capital adequacy ratios, risk mitigation strategies)
* **Procedural Guidelines** (e.g., how financial institutions should report transactions)
* **Cross-References to Other Regulations** (e.g., references to Basel III or EU directives)

To extract text from the PDFs, we used PyMuPDF libraries, which allows efficient text retrieval while maintaining the document structure. The extracted text files were saved in a new folder.

**Basic Data Stats:**

| # Basic Stats doc\_lengths = [len(word\_tokenize(doc)) for doc in documents]  print("Total number of documents:", len(documents)) print("Average document length (words):", sum(doc\_lengths) / len(doc\_lengths)) print("Max document length (words):", max(doc\_lengths)) print("Min document length (words):", min(doc\_lengths)) |
| --- |

Total number of documents: 11

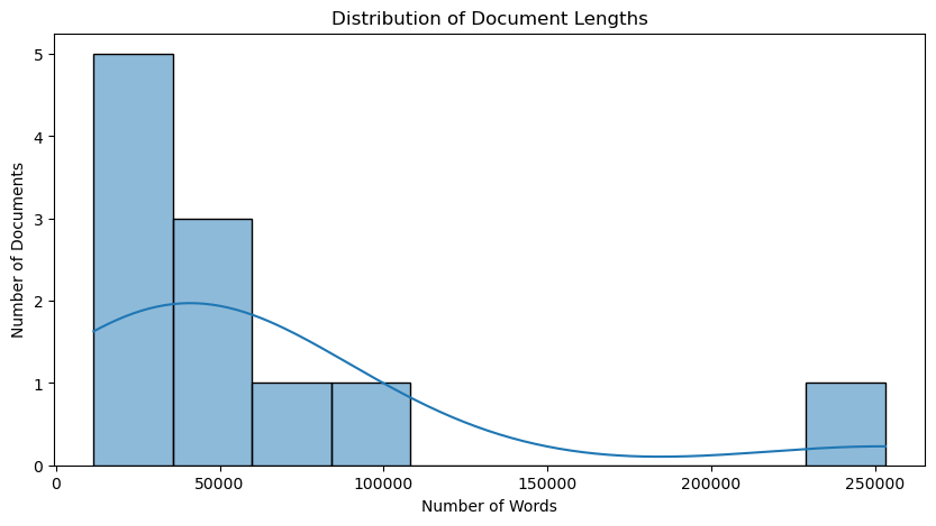
Average document length (words): 63728.90909090909

Max document length (words): 253151

Min document length (words): 11548

**Document Lengths:**

| # Plot document length distribution plt.figure(figsize=(10,5)) sns.histplot(doc\_lengths, bins=10, kde=True) plt.xlabel("Number of Words") plt.ylabel("Number of Documents") plt.title("Distribution of Document Lengths") plt.show() |
| --- |



## **Data Cleaning + Text Preprocessing**

| import os import re import string import unicodedata  def clean\_and\_advanced\_clean\_text(text):  """  Cleans and preprocesses extracted legal text by:  - Removing extra newlines and spaces  - Removing page numbers and unwanted headers  - Removing special characters and legal references  - Normalizing Unicode  - Removing tables, footnotes, and common boilerplate text  """   # Remove extra spaces and newlines  text = re.sub(r'\n+', '\n', text) # Remove multiple newlines  text = re.sub(r'\s+', ' ', text) # Collapse multiple spaces   # Remove page numbers and headers (example: "Page 12 of 50")  text = re.sub(r'Page \d+ of \d+', '', text, flags=re.IGNORECASE)   # Remove special characters (except basic punctuation)  text = text.translate(str.maketrans('', '', string.punctuation))   # Normalize Unicode (remove weird characters)  text = unicodedata.normalize("NFKD", text).encode("utf-8", "ignore").decode()   # Remove references like [Article 12] or [Section IV]  text = re.sub(r"\[.\*?\]", "", text)   # Remove table-like structures (rows of numbers, e.g., "1.2 | 3.4 | 5.6")  text = re.sub(r"(\d+(\.\d+)?\s\*\|\s\*)+\d+(\.\d+)?", "", text)   # Remove common legal boilerplate text  boilerplate\_patterns = [  r"This document is for informational purposes only.\*",  r"All rights reserved.\*",  r"Reproduction or distribution is prohibited.\*"  ]  for pattern in boilerplate\_patterns:  text = re.sub(pattern, "", text, flags=re.IGNORECASE)   # Remove footnotes (assuming they start with numbers)  text = re.sub(r"\d+\.\s.\*", "", text)   # Remove legal symbols and abbreviations (e.g., "§", "Art.", "No.")  text = re.sub(r"§|\bArt\.|\bNo\.", "", text)   # Remove excessive whitespace  text = re.sub(r"\s+", " ", text).strip()   return text  # Apply cleaning to all texts cleaned\_documents = [] data\_path = "your\_text\_files\_path\_here" for file in os.listdir(data\_path):  if file.endswith(".txt"):  with open(os.path.join(data\_path, file), "r", encoding="utf-8") as f:  raw\_text = f.read()  cleaned\_text = clean\_and\_advanced\_clean\_text(raw\_text)  cleaned\_documents.append({"filename": file, "content": cleaned\_text}) |
| --- |

| # Select a random sample sample\_doc = random.choice(cleaned\_documents) print(f"📄 File: {sample\_doc['filename']}\n") print(sample\_doc["content"][:1000]) # Show first 1000 characters |
| --- |

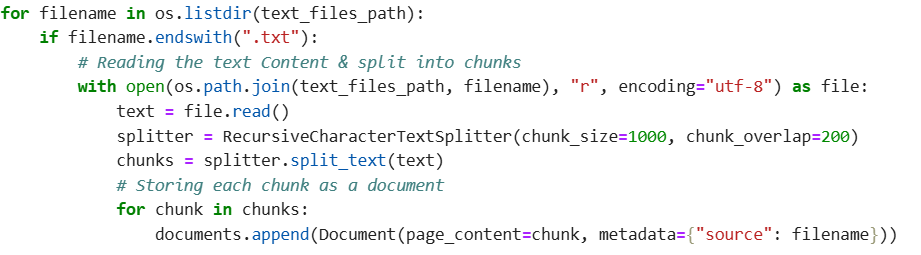
📄 File: IFR\_EURLEX.txt

I (Legislative acts) REGULATIONS REGULATION (EU) 2019/2033 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 November 2019 on the prudential requirements of investment firms and amending Regulations (EU) No 1093/2010, (EU) No 575/2013, (EU) No 600/2014 and (EU) No 806/2014 (Text with EEA relevance) THE EUROPEAN PARLIAMENT AND THE COUNCIL OF THE EUROPEAN UNION, Having regard to the Treaty on the Functioning of the European Union, and in particular Article 114 thereof, Having regard to the proposal from the European Commission, After transmission of the draft legislative act to the national parliaments, Having regard to the opinion of the European Central Bank (1), Having regard to the opinion of the European Economic and Social Committee (2), Acting in accordance with the ordinary legislative procedure (3), Whereas: (1) Robust prudential requirements are an integral part of the regulatory conditions under which financial institutions provide services within the Union. Investment firms

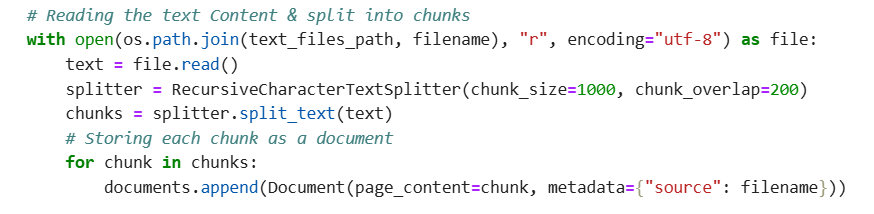
## **Embedding and Storing Chunks**

#### Embed your chunks of documents

* **Looping through all the .txt files in the folder.**



* **Creating chunks of 1000 to better optimize the model.**



* **Creating a FAISS vector database from chunks.**



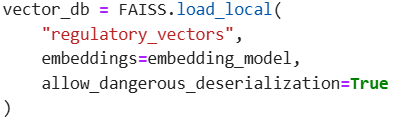
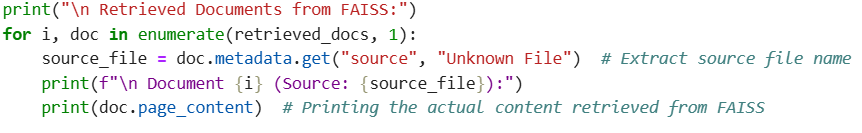
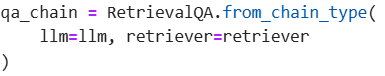
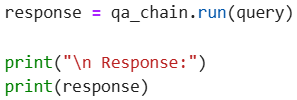
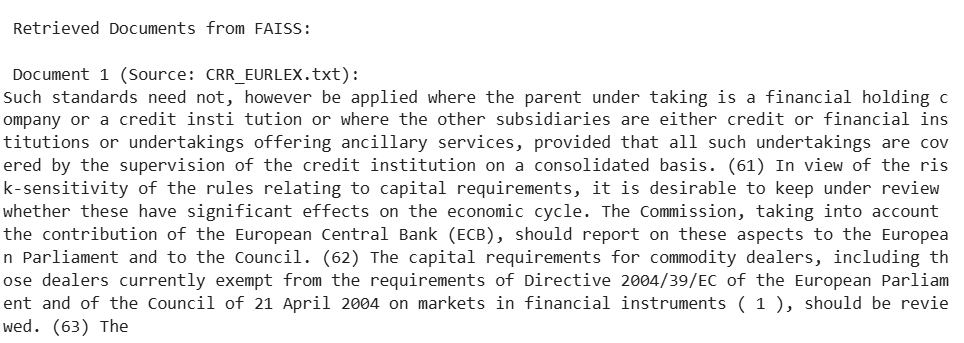
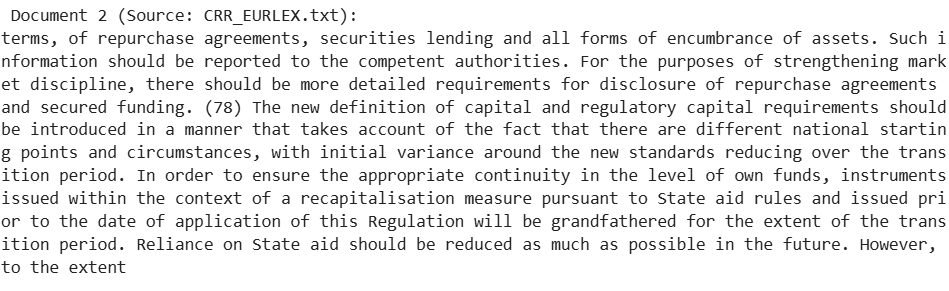
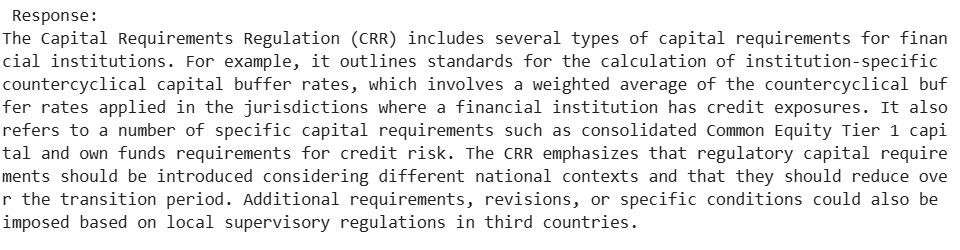
* **Saving the FAISS index locally.**



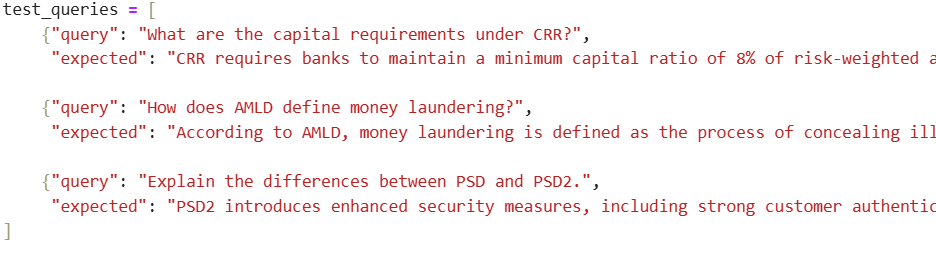
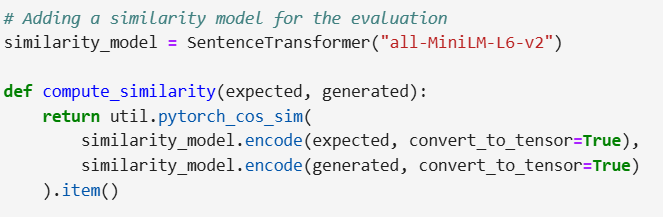
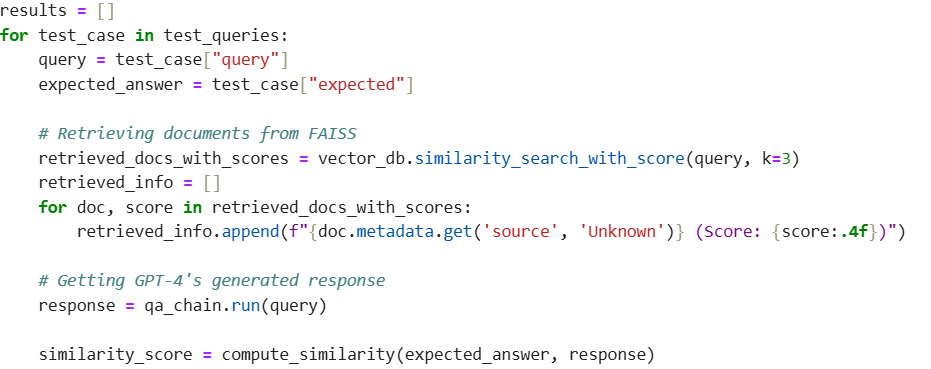
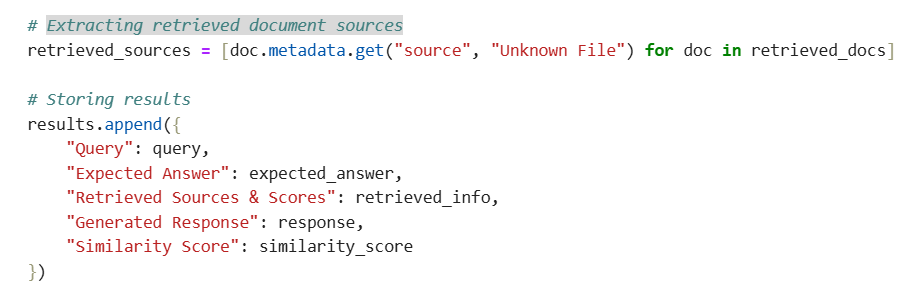
#### **Connection to Vector DB**

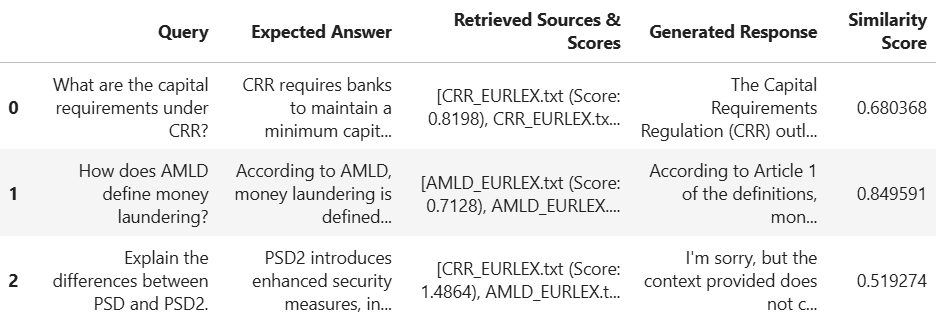
* **Setting up OpenAI key.**



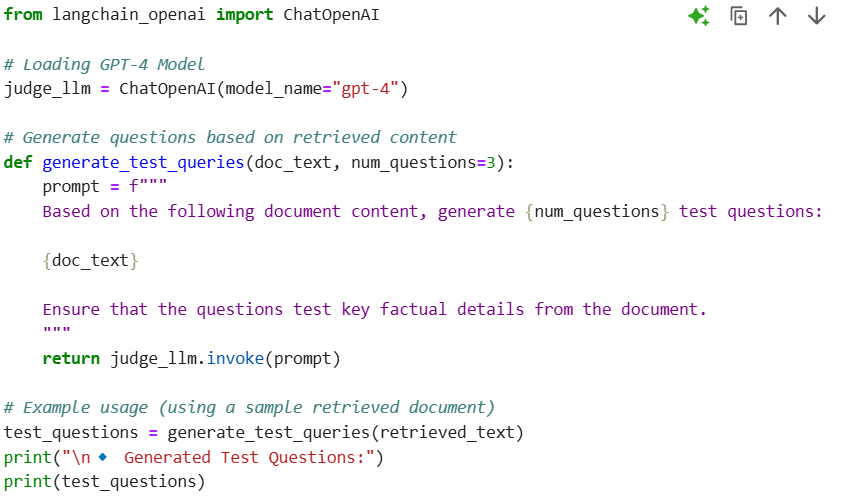
* **Loading the FAISS vector database.**
* **Creating a retriever.**
* **Asking a question.**
* **Retrieving relevant documents from FAISS.**
* **Printing received documents.**
* **Loading GPT-4.**
* **Create the RAG pipeline.**
* **Generating a final answer.**
* **Response**:
  + 
  + 
  + 

**Evaluation RAG system**

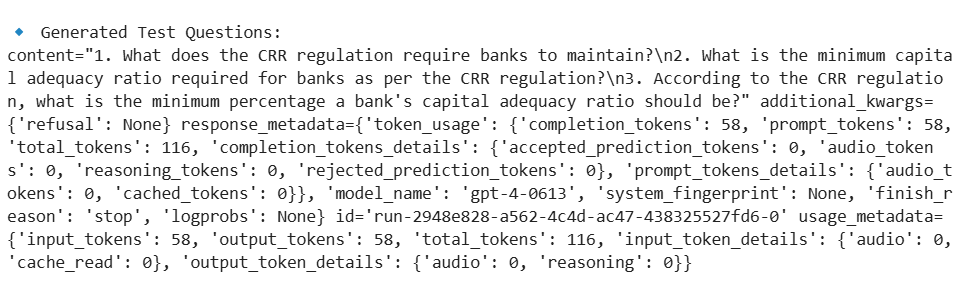
* **Generating test queries for the evaluation.**
* **Adding similarity model for evaluation.**
* **Evaluation the model.**
* **Extracting retrieved document sources and storing results.**
* **Printing evaluation results.**



**Bonus: Use an LLM as a judge to generate questions and evaluate your RAG's answers.**

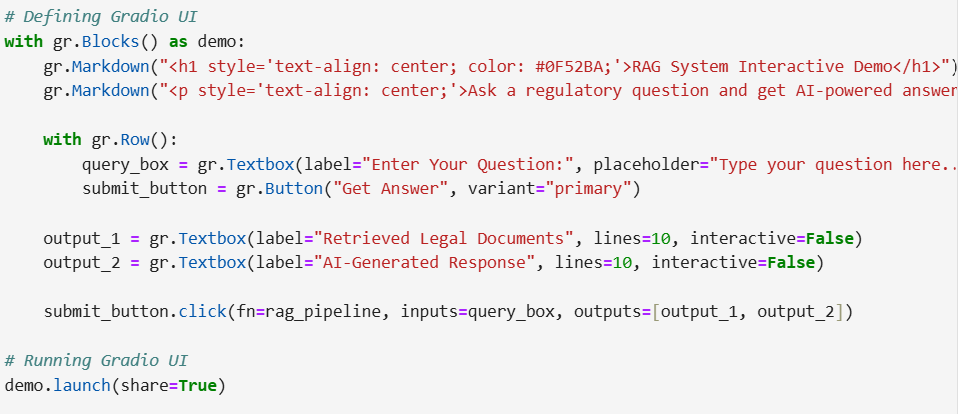


* **After running it:**



**Bonus: Provide an interactive demo of your RAG system.**





* **This would be the local url:**
  + [http://127.0.0.1:7872](http://127.0.0.1:7872/)

**Questions for the demonstration**

1. If a bank is facing financial distress, how does BRRD guide its resolution?
2. What steps must a payment service provider take to comply with PSD2 security standards?
3. What reporting obligations does a financial institution have under the CRR?
4. What happens if an investment firm does not meet IFR capital requirements?
5. Under the DGSD directive, what process must banks follow to reimburse depositors?

**Additional coding using Google Colab:**

from sentence\_transformers import SentenceTransformer

import numpy as np

# Load pre-trained embedding model

model = SentenceTransformer('all-MiniLM-L6-v2')

# Generate embeddings for the chunks

chunk\_embeddings = model.encode(chunked\_documents, show\_progress\_bar=True)

# Example: Checking the shape of embeddings

print("Shape of embeddings:", np.shape(chunk\_embeddings))

import chromadb

# Initialize ChromaDB

client = chromadb.Client()

# Try to get the existing collection

collection = client.get\_collection(name="financial\_laws\_collection")

# Store the embeddings in the collection with metadata (e.g., chunk text)

for i, embedding in enumerate(chunk\_embeddings):

collection.add(

documents=[chunked\_documents[i]], # The original chunk text

embeddings=[embedding], # The embedding for the chunk

metadatas=[{"source": text\_files[i % len(text\_files)]}], # Optionally include file metadata (e.g., source document)

ids=[str(i)] # Unique ID for each chunk

)

# Example: Querying the collection (retrieve top 5 most similar chunks)

query = "What is the recovery framework for financial institutions?"

query\_embedding = model.encode([query])[0] # Embed the query

# Perform a similarity search

results = collection.query(

query\_embeddings=[query\_embedding],

n\_results=5

)

# Print the results

for i, result in enumerate(results['documents']):

print(f"Result {i+1}:\n{result}\n")

Result 1:

['to be a governance arrangement within the meaning of Article 74 of Directive 201336EU . 2 . Competent authorities shall ensure that the institutions update their recovery plans at least annually or after a change to the legal or organisational structure of the institution , its business or its financial situation , which could have a material effect on , or necessitates a change to , the recovery plan . Competent authorities may require institutions to update their recovery plans more frequently . 3 . Recovery plans shall not assume any access to or receipt of extraordinary public financial support . 4,']

from langchain.vectorstores import Chroma

from langchain.embeddings import HuggingFaceEmbeddings

from langchain.schema import Document

# Initialize embeddings

embedding\_model = HuggingFaceEmbeddings(model\_name="all-MiniLM-L6-v2")

# Connect to ChromaDB

vectorstore = Chroma(

persist\_directory="./chroma\_db", # Specify the directory where ChromaDB is stored

embedding\_function=embedding\_model

)

# Convert stored texts into LangChain Documents

docs = [Document(page\_content=text, metadata={"source": text\_files[i % len(text\_files)]})

for i, text in enumerate(chunked\_documents)]

# Add documents to vector store

vectorstore.add\_documents(docs)

# Use LangChain's retriever to fetch relevant documents

retriever = vectorstore.as\_retriever(search\_kwargs={"k": 5}) # Retrieve top 5 relevant chunks

# Test retrieval with a sample query

query = "What is the recovery framework for financial institutions?"

retrieved\_docs = retriever.get\_relevant\_documents(query)

# Print results

for i, doc in enumerate(retrieved\_docs):

print(f"Result {i+1}:\n{doc.page\_content}\n")

**Conclusion**

#### **Strengths of the Model**

* + Effective Document Retrieval:
    - The FAISS-based retriever efficiently retrieves relevant legal documents based on the input query.
    - The use of MiniLM embeddings ensures fast and accurate similarity matching.
  + Integration of GPT-4 for Response Generation:
    - The LLM-generated responses provide contextual and well-structured answers based on retrieved documents.
    - The model leverages retrieval-augmented generation (RAG) to ground responses in factual data.
  + Structured Evaluation Approach:
    - The implementation of test queries allows for a systematic assessment of the model’s accuracy.
    - The use of similarity scoring (MiniLM Sentence Transformer) provides quantitative evaluation metrics.

#### **Areas for Improvement**

* + Enhance Document Chunking Strategy:
    - Some retrieved documents might not contain the most relevant information due to fixed chunk sizes (1000 tokens, 200 overlap).
  + Improve Query Understanding & Filtering:
    - Some queries might retrieve less relevant documents if the retriever lacks domain-specific fine-tuning.
    - Solution: Implement query re-ranking or re-weighting of retrieved documents using external metadata.
  + Optimize Response Accuracy with Fine-Tuning:
    - The model relies solely on GPT-4’s generative abilities, which can introduce hallucinations if the retrieved documents are not fully relevant.