📄 RAG Document Summarization System

This project implements a **Retrieval-Augmented Generation (RAG)** pipeline that summarizes large documents by first retrieving semantically relevant content and then generating fluent summaries using Large Language Models (LLMs).

## **🎯 Objective**

To develop a modular summarization pipeline that:

- Accepts large documents (PDF, TXT, Markdown)

- Splits them into semantically meaningful chunks

- Embeds them using transformer-based models

- Retrieves top relevant chunks via semantic search

- Generates a high-quality summary using LLMs

## **⚙️ Setup Instructions**

Using Conda (Recommended)

conda env create -f environment.yml

conda activate rag-summarization

### **🐍 Or using pip**

pip install -r requirements.txt

## **📥 How to Use**

### **1. Place Your Document**

Put any .pdf, .txt, or .md file into the sample\_documents/ directory.

### **2. Run the Pipeline**

python main.py --file sample\_documents/sample.pdf --top\_k 5

**Optional Arguments:**

* --top\_k: Number of chunks to retrieve (default: 5)

## **🧠 How It Works**

### **1. Document Ingestion**

* Handled by document\_loader.py
* Converts documents into semantic chunks

### **2. Embedding & Storage**

* Uses SentenceTransformers or OpenAI Embeddings
* Stores vectors in memory or persistent DB (e.g., FAISS, ChromaDB)

### **3. Semantic Retrieval**

* retriever.py performs top-k similarity search based on a summary query

### **4. Summary Generation**

* summarizer.py feeds retrieved context to an LLM (e.g., OpenAI GPT, Mistral, LLaMA)

## **📁 Project Structure**

Document-Summarization/

├── main.py # Entry point for running the pipeline

├── document\_loader.py # Handles document loading and chunking

├── embedding\_store.py # Embeds and stores document chunks

├── retriever.py # Retrieves top-k relevant chunks

├── summarizer.py # Generates summaries using LLM

├── config.py # Global configuration

├── run\_examples.py # Sample usage script

├── sample\_documents/ # Folder to place input documents

├── requirements.txt # Python dependencies

├── environment.yml # Conda environment config

└── README.md # Project documentation

## **📊 Sample Output**

After running the pipeline, the output will include:

* ✅ Top-k retrieved text chunks
* ✅ Final generated summary
* ✅ Optional metrics: similarity scores, token usage, latency

## **📚 Resources**

### **Datasets**

* [ArXiv Abstracts (Kaggle)](https://www.kaggle.com/datasets/Cornell-University/arxiv) - Scientific article summaries
* [CNN/DailyMail](https://huggingface.co/datasets/cnn_dailymail) - News article + summary pairs
* Local or public documents in .pdf, .txt, or .md

### **Tutorials & Guides**

* [Hugging Face RAG Tutorial](https://huggingface.co/blog/rag)
* [LangChain Quickstart](https://docs.langchain.com/docs/get_started/introduction)
* [FAISS Getting Started](https://github.com/facebookresearch/faiss/wiki/Getting-started)

### **Research Papers**

* *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* (Lewis et al., 2020)
* *Don't Pay Attention: Pay Enough! RAG Revisited* (Borgeaud et al., 2022)
* *ColBERT: Efficient and Effective Passage Retrieval*

## **🧠 Note to Interns**

This is a simplified project focusing on the fundamentals of summarization using pre-trained models and vector search. You are encouraged to be creative and modular in design. We will evaluate your effort and understanding, not just the final output.

## **✅ Submission Checklist**

* Python code with requirements.txt or environment.yml
* Sample summarization runs for at least 3 different documents
* PDF report (max 2 pages)
* This README file