Practical Assessment 7: Advanced GenAI – Retrieval-Augmented Generation (RAG)

Objective:

Implement an advanced GenAI workflow using **Retrieval-Augmented Generation (RAG)** to answer questions from a private document corpus. Build a system that indexes documents, retrieves relevant content, and generates answers with a language model.

Note:- Try to use Gemini API, as its free for personal use with rate limits

Task 1: Understanding RAG Architecture

Instructions:

- Briefly explain RAG: separation of retrieval and generation.
- Identify components: vector store, retriever, reader/generator.

Deliverables:

- Markdown cell with architecture summary and diagram (optional).
- Tools selected for each component (e.g., FAISS, OpenAI embeddings, GPT).

RAG separates retrieval and generation in AI models. First, the model retrieves relevant information from an external source (like a database or vector store), then it generates responses based on both the retrieved data and its internal knowledge.

This approach enhances accuracy and reduces hallucinations because the generated output is grounded in real information.

Task 2: Document Collection and Chunking

Instructions:

- Use your own collection (PDFs, docs, or text articles) or a sample corpus (e.g., Wikipedia, blog posts).
- Convert and split documents into chunks (approx. 200–500 words).
- Store metadata (document title, section, etc.).

Deliverables:

- Code for file reading and chunking.
- Sample of preprocessed data.

Task 3: Embedding and Vector Store Setup

Instructions:

- Use embedding model from huggingface(e.g., `sentence-transformers/all-MiniLM-L6-v2`) or gemini embedding or any ambedding model.
- Index chunks using FAISS or another vector store (e.g., ChromaDB, Pinecone).

Deliverables:

- Embedding and indexing code.
- Save/load vector index.

Task 4: Query-Based Retrieval + Generation Pipeline

Instructions:

- Input: user query
- Retrieve top-k most relevant chunks using vector similarity.
- Concatenate retrieved content as input to a **LLM** use any free tierd LLM API like groq or gemini.
- Generate an answer based on retrieved context.

Deliverables:

- Retrieval + generation pipeline code.
- Sample inputs/outputs.

Task 5: Evaluation

Instructions:

- Run the system on multiple queries.
- Use metrics like:
  - Response relevance (manual)
  - Context match
  - BLEU / ROUGE (optional for QA)

Deliverables:

- At least 5 query + result pairs.
- Commentary on whether answers are accurate/grounded.

Task 6: Web Interface or Chatbot (Optional but Encouraged)

Instructions:

- Create a simple chatbot interface using:
  - `Gradio`, `Streamlit`, or `Flask`.
- Allow users to enter questions and receive generated answers.

Deliverables:

- Running chatbot with sample queries.
- Screenshot or demo video (optional).

Task 7: GitHub + README Documentation

Instructions:

- Upload all code, models, and supporting files to GitHub.
- Include a clear `README.md` with:
  - Overview of the system.
  - Setup instructions.
  - Sample screenshots or demo video link.
  - Description of components and workflow.

Deliverables:

- GitHub repository.
- Well-documented `README.md`.