# Technical Assignment: Build a Mini LLM-Powered Question-Answering System Using RAG

## Objective

You are tasked with building a simple, functional prototype of a document-based Question-Answering (QA) system using Retrieval-Augmented Generation (RAG) powered by an open-source LLM.

This is a time-boxed challenge (4 hours) designed to evaluate your ability to:

- Think like a builder under pressure
- Design a clean RAG pipeline
- Integrate embeddings, vector search, and LLMs
- Use AI tools like ChatGPT or Copilot judiciously

---

## Provided Resources

- Input Dataset: A text document or multiple documents will be provided by Wundrsight. These may include excerpts from clinical guidelines, mental health protocols, or structured medical text.
- No need to search or scrape external data sources.

---

## Assignment Scope

### Required Components

Please implement the following in your solution:

1. Document Ingestion and Chunking
    - Load the provided text(s)
    - Split into manageable chunks for embedding (e.g., 200–500 tokens)
    - Clearly explain the chunking strategy in your code or documentation

2. Embedding and Vector Store
    - Generate embeddings using a pre-trained model (e.g., all-MiniLM-L6-v2 from Sentence Transformers)
    - Store vectors using FAISS or ChromaDB
    - Ensure retrieval of top-k similar chunks for a given query
3. Query Interface
    - Accept a text query from the user (e.g., via command line or notebook input)
    - Retrieve relevant context chunks using semantic search
4. LLM Integration
    - Use a local or free-tier accessible open-source LLM (e.g., LLaMA2, GPT4All, Mistral-7B, or any suitable HuggingFace-hosted model)
    - Use the retrieved context to generate an answer to the input query
    - You may use LangChain or other wrappers if needed, but explain your approach
5. Output
    - Display or print the generated answer
    - Include a sample input and output using this test question:

      *Give me the correct coded classification for the following diagnosis: "Recurrent depressive disorder, currently in remission"*

---

**Time Limit**

- You must complete the coding portion within 4 hours
- Use the README to note what is complete, what is skipped, and why
- You may use ChatGPT or Copilot to assist—but clearly call out where and how you used them

---

**Deliverables**

Please submit:

1. Code (in Jupyter Notebook or .py file) with clear structure and comments
2. README or explanatory note with:
    - Tools and models used
    - Where AI tools like ChatGPT/Copilot were used
    - Your design decisions and any assumptions made
    - Limitations due to time or resource constraints
3. Sample output for the provided query

---

**Evaluation Criteria**

| Area | Evaluation Focus |
|---|---|
| Functionality | End-to-end working pipeline (document → retrieval → LLM → answer) |
| Technical Design | Soundness of architectural decisions and justification |
| Use of AI Tools | Responsible and smart use of ChatGPT or Copilot |
| Code Quality | Modular, readable, and logically organized |

| | |
|---|---|
| Time Management | How well you prioritized features within time limit |
| Clarity | Assumptions and design decisions are clearly documented |

---

**Optional Bonus (if time permits)**

- Add a simple Gradio or Streamlit interface
- Allow for dynamic document uploads
- Implement basic caching for repeat queries
- Add relevance reranking logic (e.g., using Maximal Marginal Relevance)

---

**Important Notes**

- If something is unclear, assume a reasonable approach and note it in the README.
- Do not overengineer. Focus on delivering a working, clean MVP.
- Do not exceed the 4-hour coding time limit—document anything left incomplete.