**Data Science Intern role**

build a **Retrieval-Augmented Generation (RAG)-based chatbot** that can answer user queries by retrieving relevant information from a given set of documents.

**Task Details:**

* The chatbot should process and extract information from multiple document types:
  + **PDF, DOCX, CSV** → Sample files will be provided.
  + **Image and Video** → You are required to choose your own dataset and build the logic/model for processing these formats.
* The chatbot should return **accurate and relevant answers** based on the user's queries.
* Bonus points if you **do not use the LangChain framework.**

**Evaluation Criteria:**

* **Accuracy & Relevance**: How well the chatbot retrieves and generates responses.
* **Model Architecture**: Efficient implementation and logical structuring.
* **Handling of Different Formats**: Proper parsing and information extraction.
* **Creativity & Optimization**: Efficient use of techniques for better performance.

**Submission Deadline:** **25 March 2025 (Tuesday - 5 pm)**  
**Submission Format:** Code, PPT/Word

Provided -one pdf and one csv file

**TECH STACK**

 **LLM**: Open-source (LLaMA 3, Mistral, Phi-2, or Mixtral)

 **Backend**: Python (FastAPI or Flask)

 **Frontend**: **Gradio** (simple and effective for a chat interface)

 **Vector Database**: **FAISS**

 **Storage**: **Raw files + FAISS embeddings**

 **Retrieval Approach**: Hybrid (Dense + Sparse for better accuracy)

 **Processing Libraries**:

* **Text (PDF, DOCX, CSV)** → PyMuPDF, python-docx, pandas

**📌 Features List**

**🟢 FEATURE 1: Gradio Chat UI**

✅ **Chat interface** with:

* File attachment button (➕➕ for PDFs, DOCX, CSV, images, and videos).
* Text input for user queries.
* **Voice input** (speech-to-text) for querying.
* **"Suggest a Query" button** (like ChatGPT) with an explanation.

**🟢 FEATURE 2: Chat History & Search**

✅ **Maintain chat history**:

* Users should be able to **scroll back & search** past queries.
* Clear chat option

**🟢 FEATURE 3: Retrieval with Context**

✅ **Fetch relevant text + structured data**:

* Return **extracted text, tables, and images** from the source files.
* If an image is found, **show it in the response**.

**🟢 FEATURE 4: Page Number Tracking**

✅ **Display page numbers** from where the retrieved text came (for PDFs, DOCX).

**🟢 FEATURE 5: Advanced Query Handling**

✅ Users can ask for:

* **Flowcharts** 📊 (generated dynamically).
* **Tables** 📑 (structured response).
* **Image-based queries** (upload an image, and the chatbot processes it).

**STEPS**

**Steps 1:-DOCUMENT PROCESSING**

TECH USED

->fitz(PyMuPDF)

->table-pdfplumber

**1️⃣ Batch Processing of Documents (Efficiently Handling Large Corpus)**

**✅ Processed multiple files (PDF, DOCX, CSV) in a folder automatically.  
✅ Used multi-threading to speed up file processing.  
✅ Extracted text, tables, and images separately for better retrieval.  
✅ Stored images separately and linked them with metadata.**

**2️⃣ Extracting Content from Each File Type**

**✅ PDFs: Extracted text (with page numbers), tables, and images.  
✅ DOCX: Extracted text, tables, and images.  
✅ CSVs: Extracted tabular data efficiently.  
✅ Images: Stored separately and referenced in metadata.**

**3️⃣ Chunking the Extracted Data for Retrieval**

**✅ Split long text into smaller overlapping chunks (200 words per chunk).  
✅ Ensured tables are preserved as separate structured chunks.  
✅ Stored metadata (file name & page numbers) to track the source.  
✅ Saved the chunked data into chunked\_data.json for further indexing.**

**📂 Final Output of Step 1**

* **processed\_data.json → Raw extracted text, tables, and images.**
* **chunked\_data.json → Processed and chunked data for efficient retrieval.**
* **Images stored separately and linked in JSON metadata.**