AIRMAN — AI/ML Intern Technical Assignment (Document-Driven RAG Chat)

Overview

You will build an **Aviation Document AI Chat** that answers questions **strictly and only** from the aviation documents provided by AIRMAN (PPL/CPL/ATPL textbooks, SOPs, manuals).

The goal of this assignment is to evaluate your ability to:

- Design a **reliable Retrieval-Augmented Generation (RAG) pipeline**
- Prevent hallucinations and enforce grounded answers
- Provide traceable citations from source documents
- Evaluate system quality in a structured, measurable way

If an answer **cannot be supported by the provided documents**, your system **must explicitly refuse to answer**.

Time: 72 hours

Submission

- GitHub repository (code + README)
- Demo video (5–8 minutes)
- Evaluation report

Provided by AIRMAN

- 1–3 aviation PDFs (textbooks / manuals / SOPs)
- A seed list of at least **30 questions**

Required Tech Stack (preferred)

- **Language:** Python 3.10+
- **API:** FastAPI
- **RAG:** FAISS or pgvector (Postgres)
- **Embeddings:** sentence-transformers (or equivalent)
- **LLM:** Any (cloud or local), but must support grounded answers
- **Optional:** Docker

Note:
Frameworks like LangChain or LlamaIndex may be used, but **clarity of design, correctness, and grounding matter more than using a library**.

Level 1 — Mandatory (Working RAG Chat + Grounding)

Level 1 is **mandatory**. Your submission must fully satisfy this level to be considered.

A) Ingestion Pipeline

Build a document ingestion pipeline that prepares aviation PDFs for retrieval.

Your pipeline must:

1. Load the provided PDF file(s)
2. Extract text cleanly (handle page breaks and formatting issues)
3. Split the text into chunks
   - Clearly explain your chunking strategy (chunk size, overlap, rationale)
4. Generate embeddings for each chunk
5. Store embeddings in a vector index (FAISS or pgvector)

Deliverable:

- `ingest.py` script **or**
- `POST /ingest` API endpoint

B) Query → Retrieval → Answer (Chat)

Build a chat interface (API or minimal UI) where a user can ask questions and receive answers generated using retrieval.
Each response must include:

- **Answer**
- **Citations**
  - Preferably: document name + page number
  - If page numbers are not available: chunk ID + short text snippet
- **Retrieved Chunks**
  - Show at least the **top 3 retrieved chunks** when a debug flag is enabled

Hard rule (hallucination control)

If the answer **cannot be supported by the retrieved text**, the system **must respond exactly with**:

“This information is not available in the provided document(s).”

This rule is strictly enforced during evaluation.

C) Question Set + Evaluation

Question Set Creation

Create **50 questions** based on the provided aviation document(s):

- **20 simple factual questions**
  (definitions, direct lookups)
- **20 applied questions**
  (scenario-based, operational or procedural)
- **10 higher-order reasoning questions**
  (multi-step reasoning, trade-offs, conditional logic)

Evaluation

Run all 50 questions through your system and generate an evaluation report that includes:

- **Retrieval hit-rate**
  (Did the retrieved chunks actually contain the answer?)
- **Faithfulness**
  (Is the answer fully grounded in retrieved text?)
- **Hallucination rate**
  (Any unsupported claims count as hallucination)
- **Qualitative analysis**
  - 5 best answers (with explanation)
  - 5 worst answers (with explanation)

Deliverable:

- `evaluate.py`
- `report.md`

D) Minimal API

Expose at least the following endpoints:

- `POST /ingest`
- `POST /ask`
  - Returns: answer, citations, retrieved chunks
  - Supports a debug flag
- `GET /health`

Level 1 Deliverables

You must submit:

- A working RAG assistant
- Citations and refusal behavior
- 50-question evaluation set
- Evaluation report
- Demo video walkthrough

Level 2 — Optional (Advanced — Strong Preference)

Level 2 builds **directly on top of your Level 1 system**.
You are **not expected to build a separate system**.

The purpose of Level 2 is to demonstrate how you can **improve, extend, or specialize** your Level 1 RAG pipeline in a focused way.

Complete **Level 1 first**, then choose **one** of the following enhancements and integrate it into the same system.

Option 1: Hybrid Retrieval (BM25 + Vector + Reranker)

Extend the **retrieval stage** of your Level 1 pipeline.

Starting from your vector-based retrieval:

- Add BM25 keyword-based retrieval to generate candidates
- Combine BM25 and vector results
- Apply a reranker (e.g., cross-encoder) before passing context to the LLM

You must clearly show:

- Baseline metrics from **Level 1 (vector-only retrieval)**
- Improved metrics after hybrid retrieval
- Where this enhancement fits into your existing retrieval flow

---

Option 2: Query Router + Confidence Thresholding

Extend the **decision logic** of your Level 1 `/ask` flow.

- Route simple questions to a cheaper or smaller model
- Route complex questions to a stronger model
- If confidence is low, either ask a follow-up question or refuse to answer

You must explain:

- How confidence is calculated
- How routing decisions extend your Level 1 system
- When and why refusal or clarification is triggered

Option 3: GraphRAG (Mini Prototype)

Extend your Level 1 system for **structured regulatory questions**.

Using the same documents:

- Extract entities and relationships from regulatory or rule-based content
- Store them in a graph (Neo4j or in-memory)
- Combine graph traversal with vector retrieval

You must:

- Answer at least **10 regulatory questions**
- Show how graph reasoning complements your Level 1 retrieval
- Return the reasoning path (nodes and edges) used

How We Evaluate (Rubric)

Level 1 (70 points)

- Correct RAG pipeline — **15**
- Retrieval quality + chunking strategy — **15**
- Grounding + citations + refusal behavior — **20**
- Evaluation and report quality — **15**
- API usability + clean repository — **5**

Level 2 bonus (30 points)

- Demonstrated improvement with metrics — **15**
- Strong routing or graph reasoning — **10**
- Production readiness (Docker, logging, tests) — **5**

Submission Rules

- Code must run locally
- Provide `.env.example`
- Clear setup steps in README
- Demo video is mandatory
- Keep the scope tight — **reliability matters more than fancy UI**