Assignment

Duration
1 week

Background

We are exploring the capabilities of modern LLM architectures to accelerate research workflows. We've collected a set of academic papers (PDFs) on generative AI, and we need your help to build a backend system that enables intelligent question-answering over this corpus.

This system should behave like a "Chat With PDF" assistant — capable of handling **ambiguous queries**, **answering questions based on the documents**, and **performing a web search either when explicitly requested by the user** (e.g., "Search online for...") **or when the answer cannot be found in the provided PDFs**.

UI is not required — focus on core logic, modular design, and multi-agent architecture.

What you need

- **Datasets:** A list of PDF papers will be provided to you as attachments.
- **LLM Provider**
  You are free to use any LLM provider of your choice (e.g., OpenAI, Anthropic, etc.). If you encounter issues accessing a provider, feel free to reach out to us for support.
- **Web Search API:** We recommend API or mock data source of your choice.
  Here are a few **free or freemium web search APIs** you can consider:
  - Tavily → **https://www.tavily.com/**
  - DuckDuckGo Instant Answer API → **https://duckduckgo.com/**
  - SerpAPI → **https://serpapi.com/**

Deliverables

- Submit your solution as a Git repository via email. Your **README.md** should include:
  - How it works (architecture overview + brief agent descriptions)
  - How to run locally using **docker-compose**
  - How you would improve it in the future
  - Docker and **docker-compose.yml** files included

Requirements

- Programming language: **Python**
- Application server e.g. **FastAPI** or **Flask**
- Use **LLM libraries** such as LangChain, and/or LlamaIndex
- Apply **Retrieval-Augmented Generation (RAG)**
- A script to **ingest PDFs** into the database or in-memory database
- Preferred to implement **LangGraph-based multi-agent architecture**

What we expect from you
Your system should be able to:

- Answer user questions grounded in the provided PDF documents
- Handle **follow-up questions** within a single user memory session (session-based memory for one user is enough)
- Provide a **RESTful API** for:
    - Asking questions
    - Clearing memory

In addition, we want to evaluate:

- Your understanding of LangGraph and multi-agent composition
- Your ability to reason through complex queries and implement agent orchestration
- Code quality, modularity, and production-readiness

Real-World Scenarios to Handle
Your system should gracefully handle these situations:

1. **Ambiguous Questions**
   "How many examples are enough for good accuracy"
   → "Enough" is vague—needs the dataset and the accuracy target

2. **PDF-Only Queries**
   "Which prompt template gave the highest zero-shot accuracy on Spider in Zhang et al. (2024)?"
   → Zhang et al. report that SimpleDDL-MD-Chat is the top zero-shot template (65–72 % EX across models)
   What execution accuracy does davinci-codex reach on Spider with the 'Create Table + Select 3' prompt?
   → Davinci-codex attains 67 % execution accuracy on the Spider dev set with that prompt style

3. **Out-of-Scope Queries**
   "What did OpenAI release this month?"
   → The system should recognize this is not covered in the PDFs and search the web.

Hint: A robust architecture might involve a *clarification step*, followed by a *routing mechanism* to decide whether to use PDF-based retrieval or a web search.

Bonus Points

- Implement a **Clarification Agent** to detect vague or underspecified queries.
- Add a basic **evaluation system** (e.g., golden Q&A pairs, confidence scoring).

Submit—even if the edges are rough.

A working vertical slice that ingests one PDF, answers one grounded question, and clears memory proves you can learn fast, prioritise, and ship. **Reviewers value momentum and clarity of thought** far more than an unfinished grand design hidden in your local branch.

"Progress over polish" is the signal we're hunting for. Show us:

1. A running container (docker-compose up succeeds).
2. One end-to-end path.
3. Readable code & README explaining trade-offs and next steps.

If those three boxes tick, incompleteness elsewhere is acceptable. If you have any questions or need any guidance feel free to reach out to us.

Good luck and have fun!