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 autonomous 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 diagram and 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/LangGraph, and/or LlamaIndex
- Apply **Retrieval-Augmented Generation (RAG)**
- A script to **ingest PDFs** into the database or in-memory database
- Preferred and suggested 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)
- **Think/plan and decide by itself what actions to take next**. This must not be hard-coded in any way.
- Provide a **RESTful API** for:
    - Asking questions
    - Clearing memory
- A **live demo and code-walkthrough** if your submission passes.

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 e.g. no unused/duplicated code, good logging for better debugging, better code structures (separation of concerns, etc.), etc.
- For senior candidates, we will challenge you with system design questions e.g. how you can deploy this into a real production system serving millions of users.

Real-World Scenarios to Handle

Your system should gracefully handle these situations:

1. **Ambiguous Questions**
    a. "How many examples are enough for good accuracy" → "Enough" is vague—needs the dataset and the accuracy target
    b. "Tell me more about it"? → Without context/chat history, "it" is considered vague. The clarification agent needs to ask the user what "it" actually means.

2. **PDF-Only Queries**
    a. "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)
    b. 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. **Autonomous Capability** - System can think and rethink on its own
    a. "What's the state-of-the-art text-to-sql approach? And search on the web to tell me more about the authors who contributed to the approach"
    → Your multi-agent must be smart and autonomous to decide that first it needs to 1) look for the "state-of-the-art approach in text-to-sql" for the PDFs. 2) find the authors who contributed to this approach. 3) search on the web to retrieve more information about these authors.

4. **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

1. 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.
2. Hard-coded terms in agent decision making are not going to work e.g. If “current”, “latest”, etc. terms appear in the user question, route to Web Search agent.

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).
- Implement an **Autonomous Agent** capable of planning/re-planning and decide on its own what actions to take next.

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. Workable Q/A agent, even though it couldn’t answer some of the sample questions above.
4. Readable code & architecture diagram & README explaining trade-offs and next steps.

If these boxes tick, incompleteness elsewhere is acceptable. You will receive bonus points if your system could answer all the questions above. If you have any questions or need any guidance feel free to reach out to us.

Good luck and have fun!