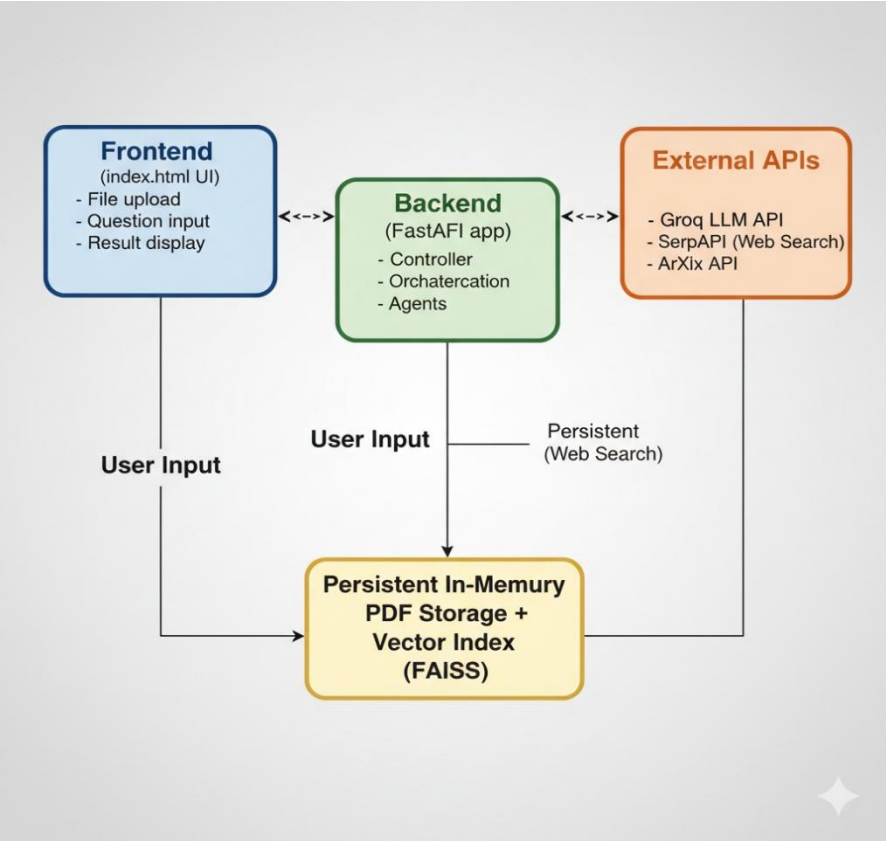
PDFy

Overview : PDFy is a powerful AI system that enables interactive question answering over domain-specific PDFs, combined with real-time web and scientific paper search. It utilizes retrieval-augmented generation (RAG) on uploaded documents alongside live web (SerpAPI) and ArXiv agents, synthesizing answers via Groq's large language model API. The system smartly orchestrates multiple agents based on query content and provides transparent logging and traceability.

Architecture :



- UI collects PDFs & queries.
- Backend orchestrates agent calls.
- PDFs are chunked, embedded, and indexed for retrieval.
- Agents: PDF RAG, Web Search, ArXiv.
- Groq's LLM synthesizes multi-source inputs into answers.
- Logs collected for traceability.

2. Agent Interfaces

PDF RAG Agent

- Input: User query + PDF chunk embeddings (top-k nearest).
- Processing: FAISS vector search over SentenceTransformer embeddings.
- Output: Relevant PDF text chunks.

Web Search Agent

- Input: User query.

- Processing: SerpAPI request (Google search) with fallback to DuckDuckGo.
- Output: Top snippet titles, summaries, and links.

ArXiv Agent

- Input: User query.
- Processing: Query via arXiv's public API for scientific papers.
- Output: Titles, abstracts, and publication links.

3. Controller Logic

Decision Rules

Basic rules for routing user queries:

- If PDF uploaded & query contains "summarize" or mentions this document → Use PDF RAG.
- If query mentions "recent papers", "arxiv", or "paper" → Use ArXiv Agent.
- If query mentions "latest news" or "recent developments" → Use Web Search Agent.
- Otherwise, combine agents reasonably (PDF RAG if available + web + arxiv).

LLM Prompt Construction

Aggregates info into a single prompt:

- User query
- Retrieved PDF chunks (top-k)
- Web results snippets or fallback text
- ArXiv paper summaries
- Agents used & decision rationale
- LLM is asked to synthesize a clear, concise answer citing sources.

---

4. Trade-offs

Benefits

- **Multi-source knowledge:** Improves accuracy using document retrieval and fresh external information.
- **RAG with FAISS:** Efficient, scalable, and fast retrieval of relevant text chunks.
- **Fallback mechanisms:** DuckDuckGo fallback improves robustness if SerpAPI rate-limit occurs.
- **Lightweight frontend:** Simple HTML/JS without heavy frameworks.
- **Structured logs:** Improves debuggability and transparency.

Limitations

- **In-memory storage:** PDF data and indices lost on server restart; no persistence yet.
- **Rate limiting:** Web API and arXiv calls can degrade under heavy load.
- **Chunk size heuristic:** Fixed chunk sizes might sometimes cut across paragraphs poorly.

- **Key management:** API keys in env; better secret management required for production.
- **No user auth:** Current system lacks authentication or privacy features.

---

5. Deployment Notes

- Environment variables in .env control API keys and limits.

- Backend built on FastAPI with Uvicorn; easily containerizable for Docker deployment.

- Frontend served as static file or embedded into backend with CORS enabled for local dev.

- Use persistent database (e.g., Redis or PostgreSQL) to store PDFs and logs for production.

- Logs stored as JSONL for scalable analysis.

- Rate limits and request sizes enforced by configuration.

- Can deploy full stack on platforms like Render, Heroku, or Huggingface Spaces with minor tweaks.

---

6. NebulaByte PDFs Generation & Usage

- NebulaByte is a curated dataset of domain-specialized PDFs (e.g., scientific articles, industry reports).

- PDFs were ingested using the upload endpoint.

- Text extraction done via PyPDF2; chunks of 500 words created.

- SentenceTransformer embeddings generated over chunks and indexed with FAISS.

- This enables semantic search to retrieve relevant content tailored to user questions.

- NebulaByte domain PDFs help test domain-generalization and relevance of RAG.

- Stored in-memory during session; for production, external DB and storage tiers recommended.

Summary

PDFSage integrates advanced retrieval and reasoning by combining RAG with multi-agent external knowledge. The system balances responsiveness with extensibility through modular agents and LLM synthesis. Logging and fallback handling increase robustness.

The architecture encourages clean separation and enables future extensions like multi-user support, advanced chunking, and persistent storage—setting a solid foundation for scalable document QA.