1. Project Overview

- A **multi-agent system** for querying PDFs, web, and Arxiv research papers.
- Integrates a **Decision Agent** to intelligently route queries and an **Answer Synthesizer Agent** to produce a clean, human-readable answer.
- Minimal frontend for PDF upload, query input, and result display.

---

2. Architecture

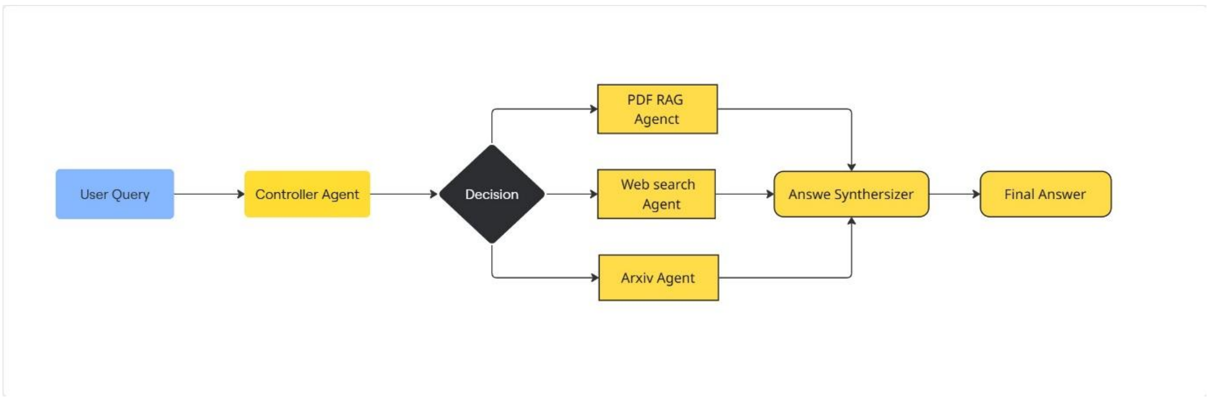
Agents:

1. **PDF RAG Agent (pdf_rag.py)**
    - Handles uploaded PDFs.
    - Uses RAG to retrieve relevant chunks of text.
2. **Web Search Agent (web_search.py)**
    - Handles queries about latest news or events.
    - Fetches snippets from web sources.
3. **Arxiv Agent (arxiv_agent.py)**
    - Handles research-oriented queries.
    - Fetches summaries from Arxiv papers.
4. **Decision Agent (decision_agent.py)**
    - Decides which agents to call based on the query and available files.
    - Returns JSON: {"agents": [...], "rationale": "..."}.
5. **Answer Synthesizer (answer_synthesizer.py)**
    - Combines snippets from multiple agents.
    - Produces a **clear, concise, factually correct** answer.

Controller (controller.py)

- Central orchestrator.
- Receives the query, asks the Decision Agent which agents to call.
- Collects results from each agent.
- Calls Answer Synthesizer for final output.

User Flow Chart



3. Controller Decision Logic

- **Checks for:**
  - File uploaded → routes to PDF agent.
  - Research keywords → routes to Arxiv agent.
  - Latest news / update keywords → routes to Web agent.
- **Default: routes to Web agent if no other matches.**

---

4. Safety & Privacy

- API keys stored in .env (never pushed to GitHub).
- No user data is logged outside the system.
- PDFs are processed locally; no external storage except temporary snippets**.**

---

5. Limitations

- PDFs with heavy images require Tesseract OCR, which is not installed on all deployment platforms.
- Gemini API usage may incur cost limits.
- Web scraping is basic; may not handle all websites.
- Arxiv summaries rely on available metadata and abstracts.

---

6. Sample Usage

1. Upload a PDF via frontend.
2. Ask a query like "Minimize this PDF" or "Latest research about LLM engineering".

3. System routes to appropriate agents and returns a synthesized answer.
4. Logs show which agents were used and the rationale.

6. Folder Structure / Deliverables

```
backend/
 ├── app/
 │   ├── agent/
 │   │   ├── pdf_rag.py
 │   │   ├── web_search.py
 │   │   ├── arxiv_agent.py
 │   │   ├── decision_agent.py
 │   │   ├── answer_synthesizer.py
 │   │   └── controller.py
 │   ├── utils/
 │   │   └── pdf_utils.py
 │   ├── main.py
 ├── uploads/
frontend/
 ├── index.html
 └── app.js
```

8. Conclusion

- Modular, easily extendable multi-agent system.
- Demonstrates RAG, web search, and decision-making orchestration.
- Deployed in Render.