Architecture: -

rag\_chatbot/

│

├── app.py # Main Streamlit or Flask app (user interface)

├── rag\_pipeline.py # Core logic: chunking, embeddings, retrieval

├── .env # Stores your OpenAI API key (not for GitHub)

├── requirements.txt # List of all dependencies

├── README.md # Project overview, setup guide, and architecture

│

├── sample\_docs/ # Folder for uploaded PDFs or test documents

│s

├── vector\_store/ # (Optional) Save FAISS index here

│

├── utils/ # Helper scripts if needed later

│ ├── preprocessing.py # (Optional) text cleaning, formatting

│ └── file\_io.py # (Optional) handling uploads or file saving

│

├── Dockerfile # (Optional) For containerization

└── .gitignore # To ignore env, .env, and large files

**1. Add Multi-Document Support**

* **Current:** Supports only a single PDF upload.
* **Improvement:** Allow users to upload multiple PDFs and ask questions across all combined content.
* **How:** Use DirectoryLoader or ConcatDocumentsChain in LangChain; manage metadata to track source document.

**✅ 2. Implement Source Attribution**

* **Current:** No explicit citation of document text or page numbers.
* **Improvement:** Highlight where the retrieved context came from (filename, page number, snippet).
* **How:** Use metadata in FAISS entries, and display them alongside responses for transparency.

**✅ 3. Add Caching or History**

* **Improvement:** Store user queries and chatbot answers to allow session-based conversation or feedback loop.
* **Bonus:** Use this to fine-tune or benchmark future improvements (retrieval quality, response time).

**✅ 4. Improve UI/UX**

* **Current:** Functional but basic Streamlit interface.
* **Suggestions:**
  + Add loading indicators and error handling (e.g., for invalid files).
  + Show retrieved document chunks as "evidence."
  + Add a model selector (e.g., choose between Gemini or GPT-4).

**✅ 5. Replace FAISS with Chroma or Hybrid Search (Optional)**

* **Why:** FAISS is great for dense retrieval but doesn’t support metadata filtering or hybrid (BM25 + dense) search.
* **Improvement:** Try Chroma or Weaviate for semantic + keyword-based retrieval (especially useful for technical PDFs).

**✅ 6. Evaluation Metrics and Logging**

* **Improvement:** Add logging of:
  + Response time
  + Token count
  + Embedding creation time
  + Retrieval hit/miss
* **Bonus:** Use this to optimize chunk size, model prompt, or retrieval logic.

**✅ 7. Add LLM Guardrails or Moderation**

* **Improvement:** Add basic checks to avoid hallucinations or inappropriate outputs.
* **Tools:** Use Guardrails AI, PromptLayer, or custom prompt filters.

**✅ 8. Dockerize the Project**

* **Improvement:** Add a Dockerfile and instructions to run the app in a container.
* **Benefit:** Easy deployment and a more professional project structure.

**✅ 9. Add Unit Tests**

* **Improvement:** Write tests for:
  + PDF loader
  + Embedding generation
  + Retrieval pipeline
* **Tools:** Use pytest, especially for backend logic.

**✅ 10. Deploy Online**

* **Improvement:** Host the project using:
  + **Render** or **Railway** for Streamlit + backend
  + Add a public link to your resume/GitHub README