**📑 AI Agent Architecture Document**

**1. Overview**

This system is a **Retrieval-Augmented Generation (RAG)–powered AI Tutor/Research Agent** designed to:

* Ingest PDFs and external sources (e.g., Semantic Scholar API).
* Extract, preprocess, and chunk text into a structured vector store.
* Retrieve relevant information based on user queries.
* Generate responses, summaries, and questions using a Groq-hosted LLM.
* Provide a conversational interface through Streamlit.

The architecture follows a **modular design** with clearly defined components for ingestion, processing, storage, retrieval, generation, and UI.

**2. System Components**

**🔹 Frontend (User Interface)**

* **Streamlit App (app.py)**
  + Handles PDF uploads.
  + Provides chatbot interface.
  + Displays summaries, key information, and generated Q&A.
  + Organizes views with expanders, tabs, and sections.

**🔹 Document Ingestion & Processing**

* **PDF Loader (pdf\_loader.py)**
  + Uses PyPDF2/pdfplumber to extract raw text.
  + Handles metadata (filename, page numbers).
* **PDF Processor (pdf\_processor.py)**
  + Cleans extracted text.
  + Splits into logical sections (abstract, methods, results, etc.).
  + Generates structured representations (sections, key\_info).
  + Summarizes papers for quick review.

**🔹 Knowledge Representation & Storage**

* **Embeddings**
  + Model: sentence-transformers/all-MiniLM-L6-v2.
  + Used for both documents and queries → ensures vector space consistency.
  + Normalized before indexing to support cosine similarity.
* **Vector Store (FAISS)**
  + Index type: IndexFlatIP (inner product → cosine similarity with normalized vectors).
  + Stores chunk embeddings and metadata (filename, section).
  + Supports efficient top-k retrieval.

**🔹 Retrieval Layer**

* **RAG System (rag\_system.py)**
  + **add\_document** → encodes text, adds vectors to FAISS.
  + **retrieve** → queries FAISS, returns top-k context chunks.
  + **chat** → integrates query + retrieved context, forwards to LLM.
  + Uses filtering logic (e.g., top-3 retrieval) to ensure non-empty context.

**🔹 Generation Layer**

* **Language Model (Groq API)**
  + Model: LLaMA-2-70B (via Groq).
  + Role: generate summaries, contextual answers, and conversational responses.
  + System prompts ensure structured, concise, and context-grounded output.
* **Question Generator (question\_generator.py)**
  + Generates comprehension questions from document text.
  + Helps in building practice Q&A for students.

**🔹 External Integrations**

* **Semantic Scholar API (semantic\_scholar\_api.py)**
  + Allows searching papers by query, citation count, or recommendations.
  + Supplements uploaded PDFs with external research context.

**3. Interaction Flow**

**Step 1: Input**

* User uploads PDF or enters query.

**Step 2: Preprocessing**

* PDF Loader extracts raw text.
* PDF Processor cleans, chunks, and stores sections.
* Embeddings generated via SentenceTransformer.
* Chunks added to FAISS index.

**Step 3: Query Handling**

* User query encoded into vector.
* FAISS retrieves top-k relevant chunks.
* RAG System composes query + context.

**Step 4: Generation**

* Groq-hosted LLM receives structured prompt:

*Question + Retrieved Context*

* LLM outputs:
  + Contextual answer (chat).
  + Summary (if summarization mode).
  + Questions (if QG mode).

**Step 5: Display**

* Streamlit shows response in chat box.
* Key info / summaries shown in expandable sections.

**4. Models Used & Justification**

| **Component** | **Model/Library** | **Reason for Choice** |
| --- | --- | --- |
| Embeddings | all-MiniLM-L6-v2 (SentenceTransformers) | Lightweight, fast, strong semantic similarity for retrieval |
| Vector Store | FAISS (IndexFlatIP) | Efficient similarity search, scalable, supports cosine similarity |
| LLM | LLaMA-2-70B via Groq | Strong generative capabilities, optimized for inference |
| PDF Processing | PyPDF2 / pdfplumber | Reliable PDF text extraction |
| UI | Streamlit | Quick prototyping, interactive frontend |
| External Search | Semantic Scholar API | Reliable scientific metadata and abstracts |

**5. Design Rationale**

* **RAG over Fine-Tuning** → Reduces compute cost, ensures knowledge freshness.
* **SentenceTransformers + FAISS** → Well-tested combo for dense retrieval.
* **Groq API** → Enables inference with very large models at low latency.
* **Streamlit** → Rapid development, modular UI, great for research assistants.
* **Tabs over nested expanders** → Fixes Streamlit hierarchy errors and keeps UI clean.