🧩 AI Agent Architecture Document

1) System Overview

An end-to-end **Summarization + RAG** agent that:

- Summarizes paragraphs and PDFs.
- Answers user questions grounded in an on-disk knowledge base.
- Runs **locally** (Hugging Face Transformers + PyTorch), optionally loading **LoRA** adapters for style/domain adaptation.
- Exposes a **Streamlit** UI.

---

2) Components

A. Presentation Layer (UI)

- **Streamlit app (`src/app.py`)**
  - Tabs: **Paragraph**, **PDF**, **Ask (RAG)**.
  - Sidebar toggle: **Use LoRA (models/lora-distilbart)**.
  - Handles file uploads and progress messaging.

B. Agent Layer

- **SummarizationAgent (`src/agent.py`)**
  - `plan(text)` → chooses **single** vs **hierarchical** strategy based on length.
  - `execute(text, plan)` → runs summarization (one-pass or chunk-then-combine).
- **RAGAgent (`src/rag_agent.py`)**

- `ingest(path)` → builds/updates KB.
- `query(question, k)` → retrieves top-K chunks, formats a concise **Context** → **Question** → **Answer** prompt, calls the summarizer.

C. Model Layer

- **Summarizer (`src/summarizer.py`)**
  - Base: `sshleifer/distilbart-cnn-12-6` (DistilBART).
  - Optional: **LoRA** adapters from `models/lora-distilbart/` (PEFT).
  - Generates summaries/answers with fixed safe caps.

D. Retrieval Layer

- **Retriever (`src/retriever.py`)**
  - **Embeddings:** `sentence-transformers/all-MiniLM-L6-v2`.
  - **Index: FAISS** (CPU) stored under `knowledge_base/`.
  - **Metadata:** `meta.jsonl` — one line per chunk with text + source.
  - `ingest_path(path)` → extract text (via `pypdf` for PDFs), chunk, embed, upsert to FAISS.
  - `search(query, k)` → return top-K chunks + scores.

E. Utilities

- **`src/utils.py`**
  - `pdf_to_text(path)` → robust text extraction.
  - `chunk_text(text)` → ~1.2k char chunks with overlap.
  - Small helpers (length, compression).

F. Data/Evals/Training

- **Training**: `training/finetune_lora.py` (LoRA adapters).
- **Evaluation**: `evaluation/eval_metrics.py` (ROUGE & compression).
- **Sample data**: `data/sample_dataset.jsonl`.

---

3) Interaction Flow

A. Summarization (Paragraph/PDF)

1. **User input** (text or PDF) → UI.
2. **Preprocess**:
    - PDF → text (`pdf_to_text`).
    - If long → `chunk_text`.
3. **Agent planning**:
    - `SummarizationAgent.plan` → {strategy: single | hierarchical}.
4. **Execution**:
    - **single**: one pass through `Summarizer`.
    - **hierarchical**: summarize chunks → join → summarize combined.
5. **Output** rendered in UI (with strategy & chunk count).

B. RAG Question Answering

1. **Ingest**:
    - User provides folder / uploads files → `RAGAgent.ingest`.
    - Text is chunked, embedded (MiniLM), and stored in FAISS with metadata.
2. **Query**:
    - User enters question → `RAGAgent.query(question, k)`.

- Retriever returns top-K chunks.

- Prompt template:

  `Answer the question using only the context.`
-
- `Context:`
- `{top-K chunks}`
-
- `Question:`
- `{user question}`
-
- `Answer:`
  -
  - `Summarizer` generates concise answer.

3. **UI** shows answer + expandable **retrieved chunks** for transparency.

---

4) Models Used

Layer	Choice	Why
Summarizer	**DistilBART (sshleifer/distilbart-cnn-12-6)**	Lightweight, summarization-tuned, runs on CPU.
Fine-tuning	**LoRA (PEFT)** on attention proj. layers (`q_proj`, `k_proj`, `v_proj`, `out_proj`)	Parameter-efficient; small adapters; easy on/off toggle.
Embeddings	**all-MiniLM-L6-v2** (Sentence Transformers)	Strong semantic similarity with small footprint, fast CPU.

Vector Store	**FAISS (CPU)**	Reliable, simple, local similarity search.
UI	**Streamlit**	Rapid prototyping; no frontend heavy lifting.

---

5) Reasons for Choices

- **Local LLM (DistilBART)**
    - No API keys, privacy-preserving, cost-free.
    - Already summarization-oriented → better starting point than generic LLMs.
- **LoRA Fine-tuning**
    - Need to adapt to **academic/student style** and improve reliability on your domain.
    - LoRA keeps compute/storage tiny (adapters are a few MB) and is easily switchable.
- **RAG**
    - Instantly incorporates new documents **without retraining**.
    - Strong grounding reduces hallucinations; transparent retrieval evidence.
- **FAISS + MiniLM**
    - Good quality-speed tradeoff on CPU; simple to deploy.
- **Agent Abstraction**
    - **SummarizationAgent** and **RAGAgent** encapsulate planning, retrieval, prompt construction, and generation → clean UI, easy testing, and future extensibility.

---

6) Non-Functional Considerations

- **Reproducibility:** deterministic chunking & capped generation; saved adapters in `models/lora-distilbart`.
- **Transparency:** "View retrieved chunks" exposes evidence used for an answer.
- **Extensibility:** swap models, add OCR, caching, guardrails, or larger vector DB without changing the UI contract.
- **Performance:** all CPU-friendly defaults; optional GPU speeds up training/inference.

---

7) Future Enhancements

- **OCR** for scanned PDFs (Tesseract).
- **Better factuality checks** (QAFactEval) for sensitive domains.
- **Caching** of embeddings and generations.
- **Multi-agent planning** (Planner ↔ Executor) with error recovery and tool usage.
- **Larger local models** (e.g., Llama-3-Instruct) if hardware allows.

---

8) Minimal Sequence Diagram (Text)

- `User -> UI(Streamlit): text/PDF/question`
- `UI -> SummarizationAgent: plan(text)`
- `SummarizationAgent -> Summarizer: generate(summary)  [or]`
- `UI -> RAGAgent: ingest(path) / query(question, k)`
- `RAGAgent -> Retriever(FAISS): search(question, k)`
- `RAGAgent -> Summarizer: generate(answer using context)`
- `Summarizer -> UI: summary/answer`

`UI -> User: display + retrieved chunks (RAG)`

-