AI Learning Copilot — Architecture & LLD Overview

1. Product Vision

AI Learning Copilot is a personal RAG-based learning assistant where users can ingest PDFs, websites, and YouTube videos and then ask questions, generate summaries, create quizzes, and test their knowledge. All answers are grounded in the user's own content and include citations.

2. Goals

- Build a production-grade RAG system using LangChain and Gemini. - Support ingestion from multiple sources. - Provide Q&A;, Summarization, Quiz, and Test modes. - Ensure answers are grounded with citations. - Include observability, cost tracking, and caching hooks from day one.

3. High-Level Architecture

The system consists of a FastAPI backend, an ingestion pipeline, a vector database, a retrieval and reranking layer, a RAG orchestration layer, and Gemini (via LangChain) as the LLM. The frontend communicates with the FastAPI backend which routes requests to either ingestion or query pipelines.

4. Logical Modules

- API Layer (FastAPI routes, auth, rate limiting) - Ingestion Module (loaders, cleaner, chunker, deduplicator) - Embeddings Module (Gemini embeddings service) - Vector Store Module (Qdrant / pgvector abstraction) - Retrieval Module (retriever, reranker, context compressor) - RAG Orchestration Module (pipelines and prompt routing) - LLM Gateway (Gemini via LangChain) - Observability Module (metrics, tracing, cost tracking) - Cache Module (Redis) - Background Workers (async ingestion jobs)

5. Folder / Package Structure

app/ ■■■ main.py ■■■ core/ (config, logging, dependencies, security) ■■■ api/ (routes_ingest.py, routes_query.py, routes_health.py) ■■■ domain/ (models.py, schemas.py) ■■■ ingestion/ (service.py, loaders/, cleaner.py, chunker.py, deduplicator.py) ■■■ embeddings/ (service.py, gemini_embeddings.py) ■■■ vectorstore/ (base.py, qdrant_store.py, pgvector_store.py) ■■■ retrieval/ (retriever.py, reranker.py, context_compressor.py) ■■■ rag/ (orchestrator.py, prompt_router.py, pipelines/) ■■■ llm/ (gemini_client.py, langchain_adapter.py) ■■■ observability/ (metrics.py, tracing.py, cost_tracker.py) ■■■ cache/ (redis_cache.py) ■■■ workers/ (ingestion_worker.py)

6. Core Domain Models

Document: id, source_type (PDF/WEB/YT), source_uri, title, created_at Chunk: id, document_id, text, embedding, metadata (page, timestamp, section) Query: text, mode (QA/SUMMARY/QUIZ/TEST), filters Answer: text, citations, tokens_used, latency_ms

7. RAG Pipelines

The system supports four main pipelines: - Q&A; Pipeline - Summary Pipeline - Quiz Pipeline - Test Pipeline Each pipeline retrieves relevant chunks, reranks them, compresses context, builds a prompt, calls Gemini via LangChain, and returns a grounded answer with citations.

8. Key Sequence Flows

Ingestion Flow: User -> /ingest -> IngestionService -> Loader -> Cleaner -> Chunker -> Deduplicator -> Embeddings -> VectorStore Query Flow: User -> /query -> RAGOrchestrator -> Retriever -> Reranker -> ContextBuilder -> PromptRouter -> Gemini -> Answer

9. Non-Functional Requirements

- Caching of embeddings and answers - Token and cost tracking - Latency measurement per step - Prompt and index versioning - Observability hooks (metrics and tracing) - Clean separation of concerns for future scaling

10. Future Roadmap

- Multi-user and multi-tenant support - Team workspaces - Evaluation pipelines and A/B testing - Model routing (cheap vs premium models) - Feedback-driven learning improvement loops