Brief Report

Introduction

This backend system is designed to implement a **Retrieval-Augmented Generation (RAG)** pipeline using **FastAPI**, with modular support for document upload, semantic search, conversational context preservation, and interview booking. Below is a summary of what each major component is used for:

Vector Database (Qdrant)

- **Purpose**: Used to store vector embeddings of document chunks.
- **Why Qdrant**:
  - High performance, scalable, production-ready vector DB.
  - Native support for **DOT** and **COSINE** similarity comparisons.

Memory Layer (Redis)

- **Purpose**: Retains conversation context for the RAG agent.
- **How**:
  - Stores past interactions per session ID.
  - Agent uses this to generate coherent and informed responses.

Metadata Store (MongoDB Atlas)

- **Purpose**: Persistently store metadata, query logs, and interview booking details.
- **Collections**:
  - file_metadata: Stores filename, chunk count, embedding model, etc.
  - conversations: Logs user questions and agent responses.
  - interview_bookings: Stores user details and timestamps of bookings.

Agent (LangChain ReAct Agent with Tools)

- **LLM Used**: gpt-4o-mini from OpenAI.
- **Behavior**:
  - Uses tools like SearchDocs to retrieve relevant text from Qdrant.
  - Maintains reasoning steps via ReAct-style prompting.
  - Can detect and execute interview booking actions

Email Notifications (SMTP)

- **Used to**:
  - Send interview booking confirmations.
- **Configured With**:
  - Environment variables for secure credentials.
- **How**:
  - After collecting name, email, date, and time from user, an email is sent confirming the booking.

Chunking Strategies

Recursive Chunking - Fixed-size splits (1000 chars, 10 overlap), Provided consistent chunk sizes. More effective for structured text or documents with many small sections. Good baseline for Q&A.

Semantic Chunking - Splits based on meaning and similarity, Resulted in variable-sized chunks, some large, some very small. For documents with mixed formatting, semantic chunking sometimes created imbalanced chunks. More computationally expensive.

Embedding Models

sentence-transformers (**MiniLM**) - Local, open-source embedding, initially intended, but issues with installation and performance on non-GPU hardware led to frequent errors.

OpenAI's text-embedding-ada-002 - Lightweight, cloud-based, 1536-dim, Chosen due to prior usage and minimal setup. Performed well, fast inference via API, and had lower cost. Results were more accurate compared to local embedding.

Similarity Search Methods

Dot Similarity Search and Cosine Similarity Search

Latency:

Both dot-product and cosine similarity search methods exhibit similar latency, with dot-product averaging **2078.7 ms** and cosine similarity slightly faster at **2057.3 ms**. The minimal difference (~21 ms) indicates that latency is largely influenced by external factors such as network communication and database overhead, rather than the similarity calculation itself.

Answer Depth:

Cosine similarity consistently provides deeper answers, covering a broader range of lessons and concepts from *Rich Dad Poor Dad*. Dot-product results tend to be more concise and focus on a narrower set of ideas.

Clarity and Coherence:
Answers generated through cosine similarity are clearer and more coherent, integrating multiple related concepts into a well-structured summary. Dot-product answers, while clear, tend to be shorter and less connected in narrative flow.

Recommendation:
Despite their comparable latency, cosine similarity is recommended when semantic accuracy, context richness, and comprehensive responses are critical—especially for applications like RAG and in-depth question answering. Dot-product search may be suitable for scenarios prioritizing slightly faster retrieval where brevity is acceptable.

.