System Design Document

Architecture, Justification, Components, and Future Work

1. Introduction

This document provides a high-level system design explanation for the PaperMind RAG-based document understanding platform. It outlines the architectural choices, underlying components, motivations, constraints, tradeoffs, and future directions. The goal is to provide a clear, structured justification of why this architecture was chosen and how each module contributes to the system.

2. Why This Architecture?

The chosen architecture follows a modular, service-oriented design composed of:

- A React frontend to manage conversations, streaming, and PDF uploads.
- A FastAPI backend responsible for routing, streaming, and orchestration.
- A background processing pipeline for embedding ingestion.
- A vector database (Qdrant) for high-dimensional search.
- An LLM layer handling query rewriting, HyDE, reranking, summarization, and final responses.

This design was selected for the following reasons:

1. **Separation of concerns:** Ingestion, retrieval, and generation are independent, reducing coupling.
2. **Scalability:** Each subsystem can scale independently (horizontal scaling for Qdrant, autoscaling for FastAPI, parallel workers for embeddings).
3. **Performance:** Background pipelines prevent blocking user-facing endpoints.
4. **Operational clarity:** Failures in ingestion or LLM calls do not affect stored vectors.
5. **Developer productivity:** Clear boundaries simplify debugging and continuous deployment.

3. Why Qdrant?

Qdrant was selected as the vector store due to:

- **Hybrid filtering:** Qdrant supports both vector similarity and metadata filtering.
- **Fast approximate search:** HNSW indexing enables sub-millisecond retrieval for millions of vectors.
- **Cloud hosting:** Managed Qdrant clusters simplify deployment and operations.
- **Payload flexibility:** Arbitrary JSON metadata can be attached to points.
- **Stability and maturity:** Qdrant Cloud provides SLAs, backups, and replication.

Alternatives such as Pinecone or Weaviate were considered but rejected due to higher cost, lower transparency, or heavier operational requirements.

4. Why HyDE?

HyDE (Hypothetical Document Embeddings) improves retrieval when user queries are under-specified, ambiguous, or require inferred context.

The system uses HyDE for the following reasons:

1. **Query expansion:** Synthetic answers produce richer embeddings.
2. **Higher recall:** HyDE generates pseudo-documents that match semantically relevant chunks.
3. **Robustness:** It compensates for noisy or incomplete user questions.
4. **Empirical performance:** Testing showed significant improvements in retrieving correct chunks.

The tradeoff is increased latency due to additional LLM calls, which is mitigated by small model usage (e.g., gpt-4o-mini).

5. System Components

5.1 Frontend (React)

The frontend provides:

- A conversational UI with streaming token display.
- A sidebar for uploading PDFs.

- Local state management for conversation history.
- Support for real-time message streaming via Fetch + ReadableStream.
- Mobile version
- A Sidebar for conversations that are named based on the first user query

5.2 Backend (FastAPI)

The backend handles:

- PDF upload routing.
- Streaming responses from OpenAI.
- RAG orchestration (query embedding, vector search, reranking, summarization).
- Conversation persistence using JSON files.

5.3 Embedding Pipeline

A fully decoupled pipeline performs:

1. PDF $\rightarrow$ JSON extraction.
2. Metadata enrichment.
3. Chunk embedding using `text-embedding-3-large`.
4. Upsert operations into Qdrant.
5. Cleanup of intermediate files.

5.4 Vector Store (Qdrant)

Qdrant stores:

- Document-level embeddings.
- Chunk-level embeddings.
- Topic centroid vectors for semantic clustering.

5.5 LLM Layer

The LLM component performs:

- Query paraphrasing.

- HyDE generation.
- Retrieval enhancement.
- LLM-based reranking.
- Context summarization.
- Final conversational answer generation.

6. Tradeoffs

6.1 Cost

- OpenAI embedding costs scale with number of chunks.
- Reranking and HyDE add LLM usage overhead.
- Qdrant Cloud introduces hosting costs.

6.2 Latency

- HyDE and reranking introduce additional milliseconds to seconds.
- Streaming mitigates perceived latency.

6.3 Complexity

A multi-stage architecture is harder to maintain but improves retrieval accuracy and robustness.

6.4 Security

- User PDFs stored only temporarily.
- No long-term retention of document content in local storage.
- API keys stored in server-side environment variables.
- Qdrant communication uses HTTPS-only.

7. Scalability

The system can scale at multiple layers:

- **Frontend:** Purely static; infinitely horizontally scalable.
- **Backend:** Stateless; can be replicated behind a load balancer.

- **Pipeline:** Parallelizable by running multiple workers per PDF batch.
- **Qdrant:** Provides built-in sharding, replication, and autoscaling.

8. Limitations

- figures or curve of a function may be poorly RAGed if complex as PDF-to-text extractor llm might find them difficult to describe
- Embedding quality depends on chunking strategy.
- HyDE increases cost and latency for every query.
- Lack of long-term conversation storage indexing may limit contextual memory.
- Summaries rely solely on retrieved context, meaning missing chunks can reduce accuracy.

9. Future Work

- Implement adaptive chunking (semantic, recursive splitting).
- Add cross-document reasoning with multi-hop retrieval.
- Introduce per-user encrypted storage for document privacy.
- Add caching for embeddings, HyDE, and reranking results.
- Support CPU-side local embedding models to reduce cost.
- Add usage analytics and quality metrics.
- Introduce model distillation for faster summarization.

10. Conclusion

This architecture balances modularity, performance, retrieval precision, and scalability. With Qdrant as the backbone for vector search and an LLM performing multi-step reasoning, the system ensures reliable and contextually grounded answers. Future enhancements will continue improving performance, cost efficiency, and retrieval accuracy.