Technical Specification Document

Project: Quick Best-Performance RAG System for Systematic Review

Name: Clifford Hepplethwaite

Email: clifford@tumpetech.com

1. Overview & Purpose

This project implements a high-performance Retrieval-Augmented Generation (RAG) system designed to support systematic review workflows by enabling researchers to query academic PDFs and receive accurate, context-aware answers with source traceability.

---

2. Architecture Overview

```
graph TD
  A[PDF Documents] --> B[LlamaIndex PDFLoader]
  B --> C[SectionNodeParser Chunking]
  C --> D[OpenAI Embeddings (text-embedding-3-large)]
  D --> E[Qdrant Cloud Vector Store]
  F[User Query] --> G[LlamaIndex Retriever (+ optional reranker)]
  E --> G
  G --> H[OpenAI GPT-4 Turbo LLM]
  H --> I[Streamlit Frontend]
```

---

3. Data Sources & Formats

- **Input data**: Academic PDF documents (systematic review papers, reports)
- **Document ingestion**: Use LlamaIndex's built-in PDFLoader which extracts text and metadata (title, author, etc.)
- **Chunking strategy**: Use `SectionNodeParser` with overlapping chunks to preserve document structure and context boundaries (e.g., Introduction, Methods, Results)

---

4. Technology Stack

Layer	Tool & Setup	Notes
Ingestion	LlamaIndex with PDFLoader	Install via `pip install llama-index`
Chunking	SectionNodeParser with overlap	Preserve academic structure for better QA

Layer	Tool & Setup	Notes
Embeddings	OpenAI `text-embedding-3-large`	Requires OpenAI API key; simple Python API call
Vector Store	Qdrant Cloud	Managed service; free tier available; API based
Retriever	LlamaIndex hybrid retriever (+ optional reranker)	Hybrid semantic + lexical search; reranker optional
LLM	OpenAI GPT-4 Turbo	Fast, high-quality text generation; requires API key
Frontend	Streamlit	`pip install streamlit`; minimal UI for querying

---

5. System Components Details

5.1 Document Ingestion & Processing

- PDFs are loaded via LlamaIndex PDFLoader.
- Extract text and metadata for each document.
- Store documents in memory for chunking.

5.2 Chunking

- Use `SectionNodeParser` to split documents by sections.
- Overlap chunks by ~20% to maintain context across boundaries.

5.3 Embedding Generation

- Use OpenAI’s embedding API `text-embedding-3-large` for semantic vector creation.
- Each chunk is embedded and stored in Qdrant.

5.4 Vector Storage & Retrieval

- Qdrant Cloud stores all vectors and supports similarity search.
- Hybrid retrieval combines vector similarity with lexical search via LlamaIndex retriever.

5.5 LLM Integration

- OpenAI GPT-4 Turbo used for answer generation.
- Queries combine retrieved context chunks into prompt with instructions for concise, citation-aware answers.

5.6 Frontend

- Streamlit app to accept user queries and display answers.
- Supports file upload for new PDFs and real-time query processing.

---

6. Integration Points

- OpenAI API for embeddings and GPT-4 Turbo calls.
- Qdrant Cloud API for vector indexing and similarity search.
- Streamlit web server for frontend.

---

7. Performance & Scalability Considerations

- Use batching for embeddings to reduce API calls.
- Qdrant indexes scale to millions of vectors with low latency.
- GPT-4 Turbo supports faster generation than base GPT-4, suitable for real-time query handling.
- Frontend designed for single-user or small team use; scale backend with FastAPI + React if needed.

---

8. Error Handling & Logging

- Handle API rate limits and retries for OpenAI calls.
- Validate PDF uploads for compatibility.
- Log user queries and errors with timestamp.
- Provide user-friendly error messages on UI.

---

9. Security & Privacy

- Store API keys securely using environment variables (`.env`).
- Transmit data over HTTPS to Qdrant and OpenAI.
- No user documents persist beyond session unless explicitly saved.

---

10. Testing Plan

- Unit test ingestion and chunking on sample PDFs.
- Integration test embedding generation and vector storage.
- End-to-end test query-answer flow in Streamlit UI.
- Test with multiple document types and large PDFs.

---

11. Deployment Plan

- Local development with Python 3.9+ environment.
- Deploy frontend on Streamlit Cloud or similar PaaS.
- Qdrant Cloud handles vector DB hosting.
- OpenAI API keys configured via environment variables.

---

12. Maintenance & Support

- Update dependencies quarterly.
- Monitor API usage and costs monthly.
- Expand to reranker model (e.g. Cohere or bge-reranker) after MVP launch.
- Add user authentication if needed for multi-user scenarios.

Appendix

Sample Environment Variables

```
OPENAI_API_KEY=your_openai_api_key_here
QDRANT_API_KEY=your_qdrant_api_key_here
```

Sample CLI Commands

```
pip install llama-index streamlit openai qdrant-client
streamlit run app.py
```
