**Advanced Multi-Modal RAG System – Project Documentation**

**By Harshal Sharma**

Generated on 2025-08-11 16:58:36Z (UTC)

# 1. Overview

This project implements a production-grade, multi-modal Retrieval-Augmented Generation (RAG) system with dynamic model adaptation. It supports ingestion of heterogeneous documents (PDF, DOCX, XLSX, CSV, TXT, and images via OCR), hybrid retrieval (dense, TF-IDF, BM25; fusion ranking), and a FastAPI backend with REST and WebSockets. An admin API enables live hot-swapping of embedding/generation/retriever backends. A Chainlit UI provides chat with real-time model switching.

# 2. Frameworks & Stack

* Language: Python 3.11+
* API: FastAPI + Uvicorn
* UI: Chainlit (chat UI with settings)
* Vector: FAISS (CPU) / In-memory; BM25 (rank-bm25); TF-IDF (scikit-learn)
* Embeddings: OpenAI / Cohere / SentenceTransformers / Hash fallback
* Generation: GPT-4o (OpenAI) / Claude (Anthropic) / Llama (Ollama) / Echo fallback
* OCR: Tesseract; Scanned PDF OCR via pdf2image + Tesseract
* Observability: Prometheus metrics; Structured logging

# 3. Project Structure

advanced\_rag/  
├── app/  
│ ├── main.py # FastAPI app factory & wiring  
│ ├── api/ # REST & WebSocket routes  
│ ├── models/ # Pydantic schemas & settings  
│ ├── pipeline/ # Ingestion & query pipeline manager  
│ ├── ingestion/ # Structure-aware chunker (extensible)  
│ ├── retrieval/ # Vector backends, sparse, BM25, fusion, service  
│ ├── index/ # Document store (persistent)  
│ ├── embeddings/ # Embedding clients & factory  
│ ├── generation/ # Generation clients & factory  
│ ├── admin/ # Admin routes & demo, dashboard HTML  
│ ├── observability/ # Logging & metrics  
│ └── security/ # Security middleware  
├── ui/chainlit/ # Chainlit UI app  
├── docs/ # Documentation & this generator  
├── tests/ # Basic tests  
├── requirements.txt # Python dependencies  
├── Dockerfile, docker-compose.yml # Containerization  
└── README.md # Quickstart (optional)

# 4. Setup & Running

Local (venv):

cd advanced\_rag  
python3 -m venv .venv  
source .venv/bin/activate  
pip install -r requirements.txt  
uvicorn app.main:app --host 127.0.0.1 --port 8000 --reload

Docker:

docker compose up --build

Chainlit UI:

export RAG\_API\_BASE="http://127.0.0.1:8000"  
chainlit run ui/chainlit/langchain\_app.py -w --host 127.0.0.1 --port 8501

# 5. Configuration

Settings are defined in app/models/config.py (pydantic-settings). You can override via environment variables or .env.

* environment: development | staging | production
* log\_level: DEBUG | INFO | WARN | ERROR
* data\_dir: ./data; index\_dir: ./data/index; uploads\_dir: ./data/uploads
* embedding\_backend: openai | cohere | sentence-transformers | hash
* embedding\_model: model string or 'hash'
* generation\_backend: openai | anthropic | ollama
* generation\_model: e.g., gpt-4o | claude-3-5-sonnet-20240620 | llama3
* retriever\_backend: faiss | chroma (in-memory stub)
* openai\_api\_key, cohere\_api\_key, anthropic\_api\_key (optional)
* admin\_api\_key: protect admin hot-swap routes (send in X-API-Key)

# 6. API Endpoints

* GET /api/health – Service health
* POST /api/ingest – Ingest documents (JSON-encoded base64)
* POST /api/ingest/upload – Upload files via multipart form
* POST /api/ingest/directory – Ingest server-side directory
* POST /api/query – Query; returns schema-compliant JSON
* WS /ws/stream – Streaming tokens with confidence updates
* GET /api/admin/dashboard – Admin UI (ingestion & hot-swap)
* POST /api/admin/hot-swap/embeddings – Swap embedding backend+model
* POST /api/admin/hot-swap/generation – Swap generation backend+model
* POST /api/admin/hot-swap/retriever – Swap retriever backend

# 7. Document Ingestion & Multi-Modal Extraction

Ingestion decodes content, persists files to ./data/uploads, extracts text depending on type, and performs structure-aware chunking. PDFs are processed page-wise preserving page numbers; scanned PDFs are OCRed via pdf2image + Tesseract.

* PDF: pypdf page text; OCR fallback per page
* DOCX: python-docx paragraphs
* XLSX: pandas read\_excel; each sheet to CSV-like text
* CSV/TXT: decoded text
* Images: Tesseract OCR

Chunk IDs encode provenance: filename\_\_p0001\_\_c0003 (PDF page/chunk).

# 8. Retrieval Architecture

Hybrid retrieval combines dense vector search with sparse methods (TF-IDF and BM25). Fusion ranking merges scores. Metadata filtering is supported. A graph reranker is included and extensible.

* Dense: EmbeddingFactory + FAISS/In-memory
* Sparse: TF-IDF (scikit-learn)
* BM25: rank-bm25
* Fusion: weighted combination of normalized scores
* Metadata filtering by source/page/mime\_type

Server-side now selects only the single most relevant document for citations and context.

# 9. Advanced Generation Pipeline

System composes a system prompt and user prompt with retrieved context. Generation uses the selected backend. Confidence is estimated dynamically from retrieval strength (top score and margin). WebSocket route supports streaming tokens.

# 10. Dynamic Model Adaptation

Admin hot-swap endpoints switch embeddings, generation, and retriever backends at runtime. Embedding/retriever swaps trigger automatic reindexing from the persistent document store. The Chainlit UI provides dropdowns and a /model command to switch generation models in real time.

# 11. Observability & Security

* Metrics: /metrics via prometheus-fastapi-instrumentator
* Logging: structured logging, level configurable
* Security: response headers (X-Content-Type-Options, X-Frame-Options, Referrer-Policy)
* Admin protection: X-API-Key header if admin\_api\_key is set

# 12. Testing

* tests/test\_retrieval.py – Index & hybrid search flow
* tests/test\_adaptation.py – Settings hot-swap state

pytest -q

# 13. Performance & Tuning

* Use SentenceTransformers locally for faster embeddings when API keys not available
* Use FAISS for larger indices; persist vectors externally for production
* Tune fusion weights and top\_k per use case
* Enable GPU for embeddings/generation where possible

# 14. Troubleshooting

* Form uploads require python-multipart installed
* Scanned PDFs require Tesseract and poppler (for pdf2image)
* Port 8000 in use: kill old Uvicorn or use a different port
* Admin 401: set ADMIN\_API\_KEY in backend and RAG\_ADMIN\_API\_KEY in UI env

# 15. Output JSON Schema

{  
 "query\_id": "uuid-string",  
 "answer": {  
 "content": "Primary response content",  
 "reasoning\_steps": ["Step 1: Analysis", "Step 2: Synthesis"],  
 "confidence": 0.87,  
 "uncertainty\_factors": ["Limited source diversity", "Temporal constraints"]  
 },  
 "citations": [  
 {  
 "document": "filename.pdf",  
 "pages": [12, 13],  
 "chunk\_id": "chunk\_001",  
 "excerpt": "Relevant text snippet with context",  
 "relevance\_score": 0.94,  
 "credibility\_score": 0.89,  
 "extraction\_method": "dense\_retrieval"  
 }  
 ],  
 "alternative\_answers": [  
 {  
 "content": "Alternative interpretation",  
 "confidence": 0.72,  
 "supporting\_citations": []  
 }  
 ],  
 "context\_analysis": {  
 "total\_chunks\_analyzed": 847,  
 "retrieval\_methods\_used": ["dense", "sparse", "graph"],  
 "cross\_document\_connections": 3,  
 "temporal\_relevance": "current"  
 },  
 "performance\_metrics": {  
 "retrieval\_latency\_ms": 245,  
 "generation\_latency\_ms": 1100,  
 "total\_response\_time\_ms": 1345,  
 "tokens\_processed": 15420,  
 "cost\_estimate\_usd": 0.034  
 },  
 "system\_metadata": {  
 "embedding\_model": "text-embedding-3-large",  
 "generation\_model": "gpt-4o",  
 "retrieval\_strategy": "hybrid\_weighted",  
 "timestamp": "2025-08-11T10:30:45Z"  
 }  
}