# Multimodal RAG Assistant (LlamaIndex + LangChain + Graph-RAG)

A production-ready multimodal Retrieval-Augmented Generation (RAG) assistant that can search across documents, images, audio, and video and answer questions with cited context. It also supports Graph-RAG enrichment with Neo4j: when your question is relational (e.g., “how is A connected to B?”), the app extracts triples from your docs and uses a knowledge graph to strengthen the answer.  
Tech stack: LlamaIndex • LangChain • ChromaDB • Neo4j • Streamlit • OpenAI/Groq • Whisper • CLIP

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## ✨ What you get

* Hybrid RAG: LlamaIndex for retrieval/indexing, LangChain for orchestration and memory
* Multimodal Search: documents, images (CLIP), audio/video (Whisper → text)
* RRF + LLM Reranking: better relevance with fusion and lightweight reranker
* Graph-RAG (Neo4j): extracts triples from text and adds relationship facts to answers
* Persistent History: user sessions stored to Neo4j
* Streamlit UI: simple, fast, and demo-ready

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 🧠 High-level workflow

• User question/media

• └─► (Optional) Transcribe audio/video → text

• └─► Detect query type (history / follow-up / new)

• └─► Rewrite follow-up into standalone question (uses conversation history)

• └─► Multimodal retrieval (LlamaIndex: text, image-CLIP, audio/video text)

• └─► RRF fusion + LLM reranking of hits

• └─► Generate answer with cited context

• └─► (If Graph-RAG ON & question is relational)

• └─► Extract triples → upsert to Neo4j → fetch supporting facts

• └─► Weave those facts into the final answer

• └─► Save Q/A + sources to Neo4j (user/session)

• └─► Show result in Streamlit (with optional graph facts expander)

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 🧰 Architecture (ASCII)

┌────────────────────────────────────┐  
 │ Streamlit │  
 │ (app.py UI) │  
 └───────────────┬─────────────────────┘  
 │  
 ┌─────────────────┴──────────────────┐  
 │ vector\_search.py (Hybrid RAG) │  
 │ - query typing & rewriting │  
 │ - multimodal retrieval (RRF) │  
 │ - LLM reranking & answer synth │  
 │ - Neo4j save & graph enrichment │  
 └───────────────┬────────────────────┘  
 │  
 ┌──────────────────────┴───────────────────────┐  
 │ llama\_index\_setup.py │  
 │ - LlamaIndex global settings (LLM, embed) │  
 │ - Connect Chroma collections (text/audio/ │  
 │ image) and build retrievers │  
 └──────────────────────┬──────────────────────┘  
 │  
 ┌──────────────┬─────────┴──────────┬──────────────┐  
 │ │ │ │  
 Document index Audio index Image index Neo4j (Graph)  
 (OpenAI emb) (OpenAI emb) (CLIP emb) - triples  
 ChromaDB ChromaDB ChromaDB - sessions/history

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 🚀 Quickstart

1) Clone & env

git clone https://github.com/<you>/Multimodal-rag-assistant--Llamaindex.git  
cd Multimodal-rag-assistant--Llamaindex  
  
# Python 3.10/3.11 recommended  
python -m venv .venv  
# Windows:  
.venv\Scripts\activate  
# Mac/Linux:  
source .venv/bin/activate  
  
pip install -r requirements.txt  
  
cp .env.example .env  
# fill in keys in .env:  
# OPENAI\_API\_KEY=...  
# GROQ\_API\_KEY=... (optional)  
# NEO4J\_URI=bolt://localhost:7687 (or Aura URI)  
# NEO4J\_USER=neo4j  
# NEO4J\_PASSWORD=your\_password

2) Put data

* Documents → data/docs/
* Images → data/images/
* Audio → data/audio/
* Video → data/video/
* (These folders are git-ignored.)

3) Build vector indexes

# Run one or more, depending on your data:  
python document\_ingestion.py  
python image\_ingestion.py  
python audio\_ingestion.py  
python video\_ingestion.py  
These scripts embed your data into ChromaDB under vectorstore/ (also git-ignored).

4) (Optional) Start Neo4j

* Local: run Neo4j Desktop or Docker
* Cloud: Neo4j Aura (copy the bolt URL, user, password to .env)

5) Run the app

streamlit run app.py

* Choose an LLM model, toggle Graph-RAG in the sidebar, and enter a User ID.
* Switch between Text / Image / Audio / Video input.
* Ask relational questions like:
* “How is OpenAI connected to Microsoft?”
* “What connects OpenAI, Azure, and Microsoft?”
* Open the “Supporting Graph Facts (Neo4j)” expander to see the extracted triples that backed your answer.

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 🗂️ File-by-file guide

**Top-level**

**app.py**

Streamlit UI. Handles file uploads, audio/video transcription (Whisper), calls the hybrid QA pipeline, toggles Graph-RAG from the sidebar, and renders answers + graph facts.

**vector\_search.py**

The core “hybrid” brain:  
- Initializes indices once and builds a MultimodalLlamaRetriever (text/audio/image).  
- Detects query type (history / follow-up / new), rewrites follow-ups.  
- Runs retrieval across modalities + RRF fusion, then LLM reranking.  
- Generates the final answer (RAG) or falls back to direct LLM.  
- Saves the interaction to Neo4j and (if enabled) calls graph enrichment:  
 Extract triples from top context, upsert to Neo4j, pull back supporting facts.

**llama\_index\_setup.py**

Centralizes all LlamaIndex setup:  
- Global Settings (LLM, embedding model, chunking)  
- Connects to ChromaDB collections  
- Builds per-modality retrievers and the MultimodalLlamaRetriever (with similarity cutoff + top-k)

**config.py**

All project configuration:  
- API keys via .env  
- Data paths (data/docs, data/images, …), vectorstore/ path  
- Embedding models (text-embedding-ada-002, CLIP model)  
- LlamaIndex chunking, top-k, thresholds  
- Neo4j URI/user/password  
- Graph-RAG toggles and thresholds (e.g., GRAPH\_RAG\_ENABLED, TRIPLE\_CONFIDENCE\_MIN)

**graph\_enrichment.py**

Triple extraction + upsert:  
- extract\_triples(text) → list of (entity1, relation, entity2, doc, chunk\_id, confidence)  
- upsert\_triples(driver, triples) → creates (Entity)-[REL]->(Entity) in Neo4j

**graph\_queries.py**

Query helpers for the knowledge graph:  
- find\_relational\_subgraph(driver, e1, e2, hops) → path search  
- format\_facts\_for\_llm(records) → compact bullet facts for the final prompt

**Ingestion scripts**

document\_ingestion.py → chunk + embed docs → Chroma documents collection  
image\_ingestion.py → embed images with CLIP → images collection  
audio\_ingestion.py → (optional) embed audio metadata/transcripts → audio collection  
video\_ingestion.py → (optional) extract audio → Whisper transcript → embed text

**Utilities / examples**

test\_neo4j.py → quickly check your Neo4j connection  
requirements.txt → pinned deps  
.env.example → safe template for secrets  
README.md → (this doc)  
LICENSE → MIT or your choice

**Data & stores (git-ignored)**

data/ → your raw files (docs/images/audio/video)  
vectorstore/ → persisted ChromaDB collections  
db/ → any local scratch DBs  
Screenshots/ → demo images for README

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 🧩 How retrieval works (under the hood)

1. Chunking + Embeddings  
- Text uses OpenAI embeddings (text-embedding-ada-002).  
- Images use CLIP embeddings (sentence-transformers/clip-ViT-B-32).  
- Audio/video → Whisper transcription → treated as text.

2. Multimodal retrieval  
- Query hits documents, audio, and images collections.  
- Results per modality are filtered by a similarity cutoff.

3. Reciprocal Rank Fusion (RRF)  
- We blend ranked lists from each modality into one stronger list.

4. LLM reranking (lightweight)  
- The top candidates are reranked using a small LLM prompt.  
- Keeps just the best 2–3 chunks for answer generation.

5. Answer synthesis  
- The selected snippets are passed into an LLM to produce a grounded answer.  
- If nothing is retrieved, we fall back to direct LLM generation.

6. Graph-RAG enrichment (optional)  
- On relational questions, we extract triples (E1-REL-E2) from the top context chunks, push to Neo4j, and fetch supporting paths/facts to include in the final prompt + UI.

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 🧪 Example prompts

* “Summarize the document that mentions OpenAI partnering with Microsoft, CLIP for images, and Whisper for audio.”
* “How is OpenAI connected to Microsoft?” (Graph-RAG shines here)
* “What connects OpenAI, Azure, and Microsoft?” (multi-hop rel)
* “What did we talk about earlier?” (history summary)
* Image upload: “What does this image show and how does it relate to our docs?”

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## ⚙️ Configuration notes

Open config.py to tweak:

* Chunking: CHUNK\_SIZE, CHUNK\_OVERLAP
* Retrieval: TOP\_K, SIMILARITY\_THRESHOLD
* Models: EMBEDDING\_MODEL, CLIP\_MODEL
* Graph-RAG:  
  - GRAPH\_RAG\_ENABLED=True (default; can be toggled in the app sidebar)  
  - TRIPLE\_CONFIDENCE\_MIN, GRAPH\_MAX\_HOPS, RELATION\_WHITELIST

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 🧪 Verifying Graph-RAG works

1. Make sure you ingested at least one doc with relationships (e.g., “OpenAI partnered with Microsoft”).

2. In the UI sidebar, ensure Enable Graph-RAG is checked.

3. Ask a relational question (“How is OpenAI connected to Microsoft?”).

4. You should see an expander “Supporting Graph Facts (Neo4j)” with items like:

5. - openai PARTNERS\_WITH microsoft (doc: rel\_test.txt, chunk: 2ad38067-..., conf: 0.82)

6. You can also run a quick Cypher in Neo4j Browser:

7. MATCH (a:Entity)-[r:REL]->(b:Entity)

8. RETURN a.name, r.label, r.source\_doc, r.chunk\_id, b.name

9. LIMIT 25;

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 🧯 Troubleshooting

* “No index found at vectorstore” → Run the ingestion scripts (document\_ingestion.py, etc.) first.
* Whisper errors → Ensure OPENAI\_API\_KEY is set and file formats are supported.
* Neo4j connection errors → Check NEO4J\_URI, NEO4J\_USER, NEO4J\_PASSWORD in .env. For Aura, use the full neo4j+s://... or bolt+s://... URI.
* Empty graph facts → Graph-RAG only triggers on relational questions and when triples are extractable above TRIPLE\_CONFIDENCE\_MIN. Try a clearer relational phrasing.
* Windows line endings warnings → Harmless; Git will normalize. You can ignore.

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 📦 Requirements (minimum)

Make sure your requirements.txt includes (or similar pinned versions):

streamlit  
python-dotenv  
llama-index  
langchain  
langchain-openai  
langchain-groq  
openai  
chromadb  
neo4j  
Pillow  
transformers

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 🔐 .gitignore essentials

.env  
\_\_pycache\_\_/  
.venv/  
data/  
vectorstore/  
db/  
Screenshots/  
\*.log

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 📝 License

MIT (or your preferred license) — see LICENSE.

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 🙌 Credits

* Built with LlamaIndex + LangChain synergy
* Graph-RAG via Neo4j
* Embeddings by OpenAI and CLIP
* UI with Streamlit