Evaluation Report and Architecture Reflections

I designed this multimodal RAG (Retrieval-Augmented Generation) system to support efficient ingestion and querying across multiple data formats—PDFs, text, images, audio, and video. From the beginning, I focused on building a modular and scalable architecture that could meet the core challenge requirements while remaining easy to modify and extend.

Evaluation Framework and Pipeline Goals

My goal was to create a reliable and efficient hybrid search pipeline that can retrieve relevant context across modalities. To evaluate that, I emphasized clear output structure, retrieval consistency, and robustness in the face of varied file types. I began with a basic test suite to check if the pipeline could:

- Accept and save files correctly from a UI.
- Run queries and return structured results.
- Extract entities and relationships.
- Log and display answers clearly.
- Render an interactive graph view of relationships.

I kept logging simple and informative so that I could confirm success/failure for each step during testing. I also structured the result output to follow a consistent format (`entities`, `relationships`, `vector_results`, etc.) to make post-processing easier.

Architecture Decisions and Reasoning

I implemented the pipeline as a single notebook UI with three key components:

1. **File Upload & Query Interface** – Built using `ipywidgets`, this UI supports uploading a file, entering a natural language query, and running the pipeline with a single button. It makes it easy to test new data inputs without modifying code.
2. **Backend Orchestration** – On button click, the uploaded file is saved locally and passed into the command line to execute the full pipeline (`main.py`). This allows decoupling of UI and backend logic, which keeps things modular.
3. **Graph Output Visualization** – I used `networkx` and `matplotlib` to visualize entity-relationship graphs extracted from the result JSON. This adds explainability and makes it easier to confirm whether the system is linking the right concepts.

I avoided using external tools like Neo4j to keep setup lightweight. Instead, I focused on building a functional prototype with minimal dependencies.

Precision and Relevance

I confirmed that the entity and relationship extraction worked as intended on diverse inputs. The pipeline consistently identified key terms and semantic relationships, even from image-based

PDF files (with embedded OCR). When image files had limited text, the vector search fallback still returned something, although context was weak—as expected.

Latency and Efficiency

Because the pipeline uses preloaded models and standard Python libraries, it ran efficiently within the notebook environment. I observed that processing a typical PDF and query took around 3–6 seconds, which is acceptable for a prototype. Heavier video/audio workloads were not tested due to time and resource constraints.

Reliability and Graceful Failure

I handled the most common failure cases:

- If no file is uploaded, the UI shows a warning.
- If the result file isn't generated, the system logs the issue.
- If there are no relationships found, the graph is skipped without breaking.

This basic error handling ensures users can still use the system even if one part fails.

Maintainability and Simplicity

The code is straightforward and easy to read. It separates UI logic, pipeline invocation, and result display clearly. The pipeline accepts common formats (.txt, .pdf, .jpg, etc.) and future upgrades (e.g., audio transcription or video tagging) can be added as modular functions.

Reflections

Looking back, I think I met most of the challenge objectives within the 72-hour limit. I focused on real functionality rather than hypothetical features. If I had more time, I would:

- Add more advanced cross-modal linking (e.g., connect image captions to PDF mentions).
- Improve ranking of results using semantic similarity or topic scores.
- Add proper logging with timestamps.
- Expand the test suite with structured eval outputs.

Overall, I'm happy with how the system turned out. It performs meaningful hybrid search, supports multimodal ingestion, and visualizes structured knowledge in an accessible way.