Write-up

Brief Summary

I built an end-to-end Retrieval-Augmented Generation (RAG) system that ingests PDF documents, stores their semantic embeddings in Supabase using pgvector, and enables grounded question-answering and long-form handbook generation through a Streamlit interface.

The system supports:

- PDF upload and ingestion
- Text extraction and chunking
- Embedding generation using a SentenceTransformer model
- Vector similarity search via pgvector
- Document-level filtering
- Grounded question answering with citations
- 20,000+ word structured handbook generation
- Downloadable Markdown output

The application ensures that responses are grounded in the uploaded documents and prevents hallucinations by enforcing strict prompt rules.

Architecture Overview

The system is composed of the following layers:

1. Ingestion Layer

- PDFs are parsed using `pdfplumber`.
- Text is split into overlapping chunks (configurable size and overlap).
- Each chunk is embedded using `sentence-transformers/all-MiniLM-L6-v2` (384-dimension vectors).
- Chunks are stored in Supabase:
  - `documents` table stores metadata.
  - `chunks` table stores content, embeddings, metadata (including page numbers), and document_id.

2. Vector Search Layer

- pgvector is enabled in Supabase.
- A SQL RPC function (`match_chunks`) performs cosine similarity search:
    - `1 - (embedding <=> query_embedding)` as similarity score
    - Optional filtering by `document_id`
- Retrieval is performed via Supabase RPC from the app backend.

3. LLM Abstraction Layer

- A common `LLMClient` interface standardizes generation.
- Supports:
    - Grok (via xAI OpenAI-compatible endpoint)
    - MockLLM fallback for testing and deterministic long-form generation

This abstraction allows swapping models without modifying orchestration logic.

4. RAG Chat Flow

1. User submits a question.
2. Query is embedded.
3. Top-k similar chunks are retrieved via pgvector.
4. Retrieved context is formatted into a grounded prompt.
5. The LLM generates an answer using only provided excerpts.
6. If no relevant excerpts are found, the system responds:
   "The uploaded PDFs don't mention this."

Citations are included using stored page metadata (e.g., `(PDF p. 2)`).

5. Long-Form Handbook Generation

The `/handbook <topic>` command triggers structured generation:

1. Generate outline (12–18 sections)
2. For each section:
    - Retrieve relevant context
    - Generate 1200–1800 words
    - Maintain rolling memory summary
3. Stop after reaching target word count (≥20,000)
4. Add final conclusion section
5. Provide downloadable Markdown output

This mimics a multi-step orchestration pipeline similar to long-form generation frameworks.

Approach Taken

Design Principles

- Clear separation of concerns (ingest, retrieve, LLM, orchestration)
- Explicit grounding to prevent hallucination
- Modular LLM interface
- SQL-level vector search for scalability
- Reproducible Supabase schema
- Clean Streamlit UX

Why pgvector + Supabase?

- Native Postgres integration
- Simple deployment model
- RPC-based similarity search
- Easy filtering by document_id
- Good balance between simplicity and production realism

Why SentenceTransformers?

- Lightweight and fast
- 384-dimension embeddings
- Reliable cosine similarity performance
- No dependency on external embedding APIs

Challenges Faced

1. Schema Mismatch Issues

Early debugging revealed column name mismatches (`doc_id` vs `document_id`) between Python and Supabase schema. This caused retrieval failures even though ingestion succeeded.

Resolution:

- Standardized on `document_id`
- Ensured RPC parameter names exactly match SQL function signature

---

2. pgvector RPC Parameter Binding

The SQL function expected `filter_doc`, but Python initially sent `filter_doc_id` or `filter_document_id`. This resulted in silent retrieval failures.

Resolution:

- Matched parameter names exactly between RPC and SQL function.

---

3. Ensuring Grounded Responses

Without strict prompt rules, LLMs may:

- Inject external knowledge
- Attach citations to unsupported claims

Resolution:

- Explicitly constrained the model to use only provided excerpts
- Added fallback behavior when no excerpts are retrieved
- Included page-based citation enforcement

---

4. Long-Form Generation Coherence

Generating 20,000+ words risks:

- Repetition
- Topic drift
- Memory loss across sections

Resolution:

- Section-by-section generation
- Rolling summary memory
- Explicit structure constraints (headings, subsections, lists)

Final Outcome

The final system successfully:

- Ingests PDFs and stores vector embeddings

- Retrieves semantically relevant context
- Produces grounded answers with citations
- Prevents hallucinations for unsupported queries
- Generates structured 20,000+ word technical handbooks
- Provides a clean and reproducible setup

The implementation demonstrates understanding of:

- RAG architecture
- Vector databases
- Embedding models
- SQL similarity search
- LLM prompt orchestration
- Long-form generation workflows
- Production-style modular design