RAG_Documentation_V1

1. The Blueprint: core.py

This file contains the RAGPipeline class, which is the **blueprint** for our powerful retrieval engine.

- **The Engine:** The core of the system is the EnsembleRetriever. It's a hybrid engine that combines two types of search for the best results:
    1. **Semantic Search (Chroma):** Finds chunks that are conceptually similar in meaning.
    2. **Lexical Search (BM25):** Finds chunks that contain the exact keywords from the query.
- **Chunking Strategy:** We use RecursiveCharacterTextSplitter with a prioritized list of separators. This is our "advanced chunking" method that attempts to split text along natural boundaries (like paragraphs and sentences) to keep the chunks coherent.

2. The Control Panel: fetch.py

This file is the simple "front door" to the entire system.

- **RAG_CONFIG:** This dictionary is the main **control panel**. All settings (model names, file paths, chunk sizes) are managed here for easy experimentation.
- **Singleton Pattern:** The get_rag_pipeline() function acts as the machine's main power switch. It ensures the slow, expensive setup process runs **only once**, saving time and memory on subsequent calls.

3. Evaluation: rag_evaluation.py

This script's only job is to test our machine. It calculates several metrics, including **Precision, Recall, and Mean Reciprocal Rank (MRR)**. It has two modes:

- **'simple' mode:** A fast, mathematical check using cosine_similarity. This is a valid and efficient metric because our embedding model (all-MiniLM-L6-v2) produces normalized vectors, where cosine similarity and L2 distance are mathematically equivalent for ranking.
- **'llm_judge' mode:** A slow but deep analysis where another AI acts as a judge to determine if the retrieved text factually supports the ground truth (Context Recall).

4. Upgrading to V2: The Roadmap

This V1 is a strong foundation. Future upgrades include:

- **Speed:** Replace the Chroma database with a **FAISS** index for significantly faster vector search.
- **Accuracy:** Add a **Re-ranker** model after the retriever to get an even more precise final list of chunks. This will likely improve our low Context Recall score.
- **Features:** Modify the search function to preserve and return **metadata** to enable citations.