Deduplicated PDF Ingestion Pipeline for Qdrant

Problem Description

The task is to build a Python-based pipeline that ingests a new batch of PDF documents (Set A) into a Qdrant vector database already containing another batch (Set B), **only adding documents not already present**. Each PDF is parsed to extract its full text (e.g. via LlamaIndex's PDF reader) and then converted into a vector embedding using the *mixedbread-ai/mxbai-embed-large-v1* model. Before adding a document, the pipeline should check for duplicates by comparing **both the filename and the exact text content** to the existing entries in Qdrant. A common strategy is to compute a secure hash (e.g. SHA-256) of the document text and store it as part of the document's metadata. During ingestion, if no matching filename *and* content hash exists, the pipeline should upsert the new vector into Qdrant along with metadata (filename, file size, upload date, etc.), otherwise skip the document. This ensures each unique document is stored only once.

Requirements and Deliverables

- **Environment & Tools:** Python 3.x; libraries including LlamaIndex (for PDF parsing), the sentence-transformers package, and the Qdrant Python client.

- **Embedding Model:** Use the mixedbread-ai/mxbai-embed-large-v1 sentence-transformer model from HuggingFace for converting text to embeddings

- **Vector Database:** Use Qdrant as the vector store. Ensure a Qdrant collection exists (or create it) with the correct vector dimensionality (e.g. 512 for the MixedBread model) and a chosen distance metric (e.g. cosine).

- **Deduplication:** Check each new PDF against existing entries in Qdrant using both filename and text content.

- QDrant URL:
  https://99b8a330-4dc4-44b0-af49-2e75f98b5ece.us-east-1-0.aws.cloud.qdrant.io:6333/dashboard#/collections/v6_rag
  API Key:
  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhY2Nlc3MiOiJtIiwiZXhwIjoxNzYwNjAyOTYzfQ.Emm6hBZ6F6t7ClLL_G874LO8bFfkEouDrnU_a4bIFQw

- **Deliverables:** A script or set of scripts implementing the pipeline, a requirements.txt, a README.md with usage instructions, and example tests. The code should be

well-organized into modules and include comments. Example deliverables might include sample PDFs and expected results of the ingestion (counts of new vs. duplicate documents, etc.).

- **Metadata:** For each new document ingested, include metadata such as filename, file_size (in bytes), and an upload_date timestamp. LlamaIndex's readers automatically populate some metadata fields (e.g. file name and size).
- We currently generate and store each chunk with a payload similar to the following example:
    - {
    - "file_name": "1211523415_20130607_CCS1_Ambient_Falloff_Test_Report",
    - "extension": ".pdf",
    - "category": "oil_gas",
    - "created_at": "2024-12-31",
    - "chunk_num": "551",
    - "highlighted_chunk": "1 | ... | 3020.688 3020.",
    - "_node_content": "{...}", // serialized internal node structure with nested metadata and relationships
    - "_node_type": "TextNode",
    - "document_id": "1211523415_20130607_CCS1_Ambient_Falloff_Test_Report_551",
    - }
- **Integration Tests:** Create a small Set B with a couple of documents already in Qdrant, then run the pipeline on Set A that includes a mix of new and duplicate PDFs. Verify that after ingestion, Qdrant's collection size increases only by the number of truly new documents.
- **Error Handling:** Test behavior when encountering corrupted PDFs, or Qdrant connection failures. The pipeline should handle exceptions gracefully (e.g. skipping problematic files with a logged warning).