Workflow — Advanced Document Analysis

Page 1 — Upload page

---

1. User uploads a document
2. Document is processed by Unstructured
    1. All the elements are extracted from the document
    2. Of these elements, only the text data is extracted for usage
3. Text data (from above elements) are compiled into one entire paragraph of a chunk
    - Eg: Texts ["Hello", "I am"] from 2 separate elements are combined together into "Hello I am"
4. The compiled text data is then chunked semantically, with a specific chunk and overlap size
    1. Chunk size decides how big a chunk can be
    2. Overlap decides the overlap of words between two chunks
    3. Semantic chunking accounts for endings of sentences, there will be chunks which do not meet the chunk size
        - Part of a previous chunk will be present in the current chunk
    4. Eg:
        - Text: "Hello, I am Amrit Sutradhar. I love playing Dark Souls 3."
        - Chunk Size: 4
        - Overlap: 3
        - Output Chunks:
            - Hello, I am Amrit
            - I am Amrit Sutradhar.
            - I love playing Dark
            - playing Dark Souls 3.
5. These chunks are provided a unique identifier based on its content to prevent duplicate chunks
6. The chunks with unique identifiers are now passed to the instantiated Vector Database (ChromaDB) Collection
    1. This Collection has a Custom Embedding Function (Embedder class) which uses `nomic-embed-text` model to create embeddings for the chunks
    2. This Collection also uses `cosine-similarity` as its distance determining function
    3. Each chunk is stored in the Collection, with the information being
        - Chunk text data

    - Chunk identifier
    - Chunk embeddings
    - Chunk metadata (optional dictionary)
7. These Embeddings are now called **Documents**

Page 2 — Chat window

---

1. The user sees a chat window on their screen, and can ask anything related to the document
2. The user query is passed to the Vector Database (ChromaDB) Collection
    1. The query is converted into an embedding
    2. `n` number of documents (embedded chunks from `6.` above) which are *closest* to the query are retrieved from the Collection
3. Out of these retrieved documents, ones which are under a particular maximum distance threshold (relevant documents) are used for further processing
4. A **Structured Prompt** is engineered to instruct our **Language Model** (`llama3.1:8b-instruct-q6_K`) to answer the user's query with respect to the relevant documents as context for the query. The Structure is as follows:
    1. Context -> The retrieved documents which are under the maximum threshold
    2. Query -> The user provided query
5. This **Structured Prompt** is passed to the Language Model, and the model then generates a response based on it
    - This response can be a Regular or a Streaming response
6. This response is then shown to the user.

Page 2 (Irregular) — Error page

---

1. This page pops up if the document shared by the user has no element with any text content inside it.
2. It prompts the user to send another document instead as no text could be retrieved from their original document.

Key information

---

1. Models (running locally on Ollama) used
    1. Language Model: `llama3.1:8b-instruct-q6_K`
        - It is a 6bit Quantized model for the original `llama3.1:8b` with 16bit/32bit floating-point
        - It is much smaller compared to the original, and comes with a chance for inaccuracy
        - It is an `instruct` model which is primarily trained to perform *question-answering* or *instruction-based* tasks more effectively
    2. Embedding Model: `nomic-embed-text` (137M parameters, 16bit)
        - Only generates Embeddings for text
            - These Embeddings for one particular chunk generated is meaningful, and interlinked semantically with each word in the sentence
        - It has an 8k context window
2. Python Libraries
    1. Unstructured : Used to extract elements from documents (any document type)
        - Can be done locally on our machine (with further dependencies), or use the unstructured API
    2. Streamlit : To build the frontend for our project
    3. Ollama : To make use of open-source models locally on our machine
        - Language and Embedding models are used from here
    4. ChromaDB : To store the Embeddings generated by the models, and perform vector search using `cosine-similarity`
    5. SemanticTextSplitter : To chunk the combined element texts semantically, respecting to sentence endings and allowing overlap between chunks
        - Embeddings are created for these chunks