Documentation for PDF and Text Chunking with Chroma

Aim

The aim of this project is to provide a reusable framework for building a **Basic Retrieval-Augmented Generation (RAG)** system. It processes documents, stores them in a vector database, and enables seamless retrieval and question-answering functionality.

Objective

1. Create a structured pipeline for loading and preprocessing document data.
2. Implement text chunking methods (Recursive, Semantic, Markdown) for efficient data organization.
3. Leverage Chroma as a vector database for storing and retrieving document embeddings.
4. Provide a user-friendly interface for querying document data using state-of-the-art AI models.
5. Enable easy integration into future projects by developers for scalable RAG systems.

Scope

This framework is designed for developers and teams working on projects that require:

- Integration of a document-based RAG system for seamless retrieval and AI-driven query resolution.
- Handling large documents with unstructured data for diverse domains.
- A modular and scalable pipeline for applications in customer support, legal document analysis, and educational content management.
- Flexibility for adaptation to custom requirements or advanced RAG workflows.

This ensures future developers can integrate or expand the system effortlessly for various use cases.

Key Functionalities

This script is designed to perform the following tasks:

1. **Load and preprocess a PDF document.**
2. **Split the document into chunks** using different methods: recursive, semantic, and markdown header-based.
3. **Store the chunks** in a Chroma vector store.
4. **Retrieve relevant chunks** based on a user query using a machine learning model.
5. **Generate answers** by querying the Chroma vector store with the user's input.

Steps

1. Document Loading

- The document is loaded using the `PdfReader` class from the `PyPDF2` library.
- The entire text is extracted from the PDF and stored as a single string.
- If there's an error in reading the PDF, an exception will be raised.

2. Text Preprocessing

- **Remove Punctuation:** Removes all punctuation from the document using Python's `string.punctuation`.
- **Remove Extra Spaces:** Cleans any extra spaces after removing punctuation to ensure smooth processing.

3. Text Chunking

- **Recursive Splitting:**

    - Uses `RecursiveCharacterTextSplitter` to split the document into smaller chunks.
    - Each chunk is a fixed size with a small overlap to ensure continuity.
- **Semantic Splitting:**
    - Splits the document based on semantic meaning using the `SemanticChunker`.
- **Markdown Header Splitting:**
    - Splits the document at specific Markdown headers (e.g., H1, H2, H3) to preserve structure.

4. Storing in Chroma

- **Chroma as Vector Store:**
    - Chroma is used to save document chunks with embeddings generated by `OpenAIEmbeddings`.
    - Chunks are stored in separate collections based on the chunking method used (e.g., recursive, markdown, semantic).

5. Retrieval and Q&A

- The user selects a Chroma collection (recursive, markdown, or semantic) and enters a query.
- Using the chosen retriever, relevant documents are retrieved.
- A **question-answering chain** (`RetrievalQA`) is executed with the `ChatOpenAI` model to generate answers based on the retrieved chunks.

6. User Interaction

- The script prompts the user to:
    - Select a Chroma collection.
    - Enter a query.
- Based on the query, the script retrieves relevant document chunks and provides answers.

Setup Instructions

1. **Install Dependencies:**

Run the following command to install the required libraries:

```
pip install PyPDF2 langchain langchain-openai langchain-experimental fpdf dotenv
```

2. **Add OpenAI API Key:**

Create a `.env` file in your project directory and add your OpenAI API key:

```
OPENAI_API_KEY=your_openai_api_key
```

3. **Run the Script:**

Execute the script using Python:

```
python your_script.py
```

Conclusion

This script provides a robust framework for extracting, preprocessing, chunking, and querying text data from PDFs using Chroma and OpenAI embeddings. It's an ideal solution for building a Retrieval-Augmented Generation (RAG) system to answer questions based on large documents.