Project Overview

This project implements a sophisticated **Retrieve-Augment-Generate (RAG)** system designed to process, index, and retrieve both text and image data efficiently. Leveraging advanced models and techniques, the system facilitates seamless interaction through a web interface, enabling users to train the model with new data and test queries against the indexed information.

File Structure and Descriptions

Below is an overview of the project's core files, their functionalities, and how they interconnect within the system:

1. query_processor.py

- Purpose: Handles the main query processing pipeline. It expands user queries, retrieves relevant documents from the Chroma database, compresses context, and generates responses using the Ollama language model.

Key Functions:

- main(): Initiates the query processing.
- query_rag(query_text: str): Orchestrates the RAG pipeline steps.

2. document_preprocessor.py

- Purpose: Preprocesses documents by extracting text and images from PDF files. It also generates descriptions for images using the Ollama model.

Key Functions:

- preprocess(file_path): Extracts and processes text and image data.
- load_documents(data_path): Loads and converts preprocessed data into Document objects with unique IDs.

3. text_chunker.py

- Purpose: Splits large text documents into manageable chunks to facilitate efficient indexing and retrieval.

Key Components:

- RecursiveCharacterTextSplitter: Configured with specific chunk size and overlap parameters.
- split_documents(documents): Splits documents into chunks.

4. web_interface.py

- Purpose: Provides a Gradio-based web interface for interacting with the RAG system. Users can train the model with new data and test queries through this interface.

Key Components:

- WebInterface class: Encapsulates functionalities like resetting the database, training the model, and testing queries.
- create_interface(): Builds the Gradio interface with "Train" and "Test" tabs.

5. db_text_ingestion.py

- Purpose: Manages the ingestion of text data into the Chroma database. It handles preprocessing, chunking, and adding documents to the database.

Key Functions:

- main(data_path): Entry point for processing and adding text data.
- preprocess_pdf(file_path): Extracts and structures text from PDFs.
- add_to_chroma(chunks): Adds processed chunks to the Chroma database.
- clear_database(): Clears the existing Chroma database.

6. db_image_ingestion.py

- Purpose: Handles the ingestion of image data into the Chroma database. It processes images, generates embeddings, and stores them.

Key Components:

- DatabaseOperations class: Includes methods to clear the database, process images, and add to Chroma.
- process_images(): Processes image files and generates Document objects with embeddings.

7. text_embeddings.py

- Purpose: Provides functions to generate text embeddings using the Ollama model.

Key Function:

- get_embedding_function(): Returns an instance of OllamaEmbeddings for generating text embeddings.

8. query_expander.py

- Purpose: Enhances user queries to improve retrieval accuracy by making them more specific and detailed.

Key Function:

- rewrite_query(original_query): Uses the Ollama LLM to reformulate queries.

9. prompt_templates.py

- Purpose: Defines prompt templates for few-shot learning and image description generation.

Key Components:

- FEW_SHOT_PROMPT_TEMPLATE: Template for generating answers based on context and questions.
- image_description_prompt: Template for summarizing images for retrieval.

10. rag_evaluator.py

- Purpose: Evaluates the performance of the RAG system using metrics like Recall@K, Mean Average Precision (MAP), and Exact Match.

Key Components:

- GROUND_TRUTH: Dataset containing queries, expected answers, and relevant document IDs.

- Evaluation functions: recall_at_k_score, mean_average_precision, exact_match_score.
- evaluate_rag_model(ground_truth): Orchestrates the evaluation process.

11. clip_embedder.py

- Purpose: Generates embeddings for images and text using the CLIP model.

Key Functions:

- get_image_embedding(image_path): Generates image embeddings.
- get_text_embedding(text): Generates text embeddings.
- get_embedding_function(data_type): Returns the appropriate embedding function based on data type.

12. image_embedder.py

- Purpose: Provides advanced image embedding functionalities using the Vision Transformer (ViT) model from Hugging Face.

Key Components:

- ImageEmbedder class: Handles image preprocessing and embedding generation.
- get_embedding_function(): Returns an instance of ImageEmbedder.

13. image_embedding_generator.py

- Purpose: Optimizes the generation of image embeddings with batch processing and enhanced error handling using the ViT model.

Key Components:

- ImageEmbeddingGenerator class: Manages batch embedding generation.
- Methods: generate_embeddings, _process_batch, _load_and_preprocess_image, _batch_generator.

14. config.py

- Purpose: Centralizes configuration settings, defining paths used across the project.

Key Variables:

- CHROMA_PATH: Path to the Chroma database.
- DATA_PATH: Path to the data directory.

15. document_reranker.py

- Purpose: Enhances the relevance of retrieved documents by reranking them based on their relevance scores to the user's query using the Ollama model.

Key Functions:

- initialize_ollama(model_name): Initializes the Ollama model.
- get_relevance_score_ollama(query, document, model_name): Retrieves relevance scores from the model.
- rerank_documents(query, doc_ids, chroma_db_path, model_name): Reranks documents based on relevance scores.

Models and Techniques Used

The project utilizes a combination of specialized models and techniques to ensure efficient data processing, retrieval, and response generation:

 Language Models:

- **OllamaLLM:** Utilized for query expansion, image description generation, and response generation.
- **Mistral Model:** Specifically used in query_expander.py for rewriting queries.
- **Embedding Models:**
- **OllamaEmbeddings:** Generates text embeddings for indexing and retrieval.
- **CLIP (ViT-B/32):** Generates embeddings for both images and text, facilitating multimodal retrieval.
- **Vision Transformer (ViT):** Employed in image_embedder.py and image_embedding_generator.py for advanced image feature extraction.

 Database:

- **Chroma:** Serves as the vector database for storing and retrieving embeddings efficiently.

 Web Interface:

- **Gradio:** Provides an interactive web interface for users to train the model with new data and test queries.

 Document Processing:

- **PyMuPDF (fitz):** Extracts text and images from PDF documents.
- **PIL (Python Imaging Library):** Handles image processing tasks.

 Text Processing:

- **RecursiveCharacterTextSplitter:** Splits large text documents into smaller chunks for better indexing and retrieval.

 Evaluation Metrics:

- **Recall@K:** Measures the proportion of relevant documents retrieved in the top K results.
- **Mean Average Precision (MAP):** Evaluates the precision across multiple recall levels.
- **Exact Match:** Checks if the generated answer exactly matches the ground truth.

Data Flow and Interactions

The components of the project are interconnected in a seamless pipeline to facilitate efficient data processing and retrieval:

 Data Ingestion:

- **Text and Images:** Users upload PDF files and image files through the Gradio web interface (web_interface.py).
- **Preprocessing:** document_preprocessor.py extracts text and images from PDFs. Images are saved and described using OllamaLLM.

 Embedding Generation:

- Text data is embedded using OllamaEmbeddings from text_embeddings.py.
- Image data is embedded
  using CLIP via image_embedder.py or image_embedding_generator.py for more advanced embeddings.

 Database Management:

- **Chroma Database:** Processed and embedded documents are stored in the Chroma database (db_text_ingestion.py, db_image_ingestion.py).
- **Indexing:** Documents are uniquely identified and indexed for efficient retrieval.

 Query Processing:

- **User Query:** Through the web interface, users input queries (web_interface.py).
- **Query Expansion:** query_expander.py reformulates the query for improved retrieval.

 Retrieval:

- Expanded queries are used to fetch relevant documents from Chroma (query_processor.py).
- Contextual compression is applied to refine the retrieved information.
- **Reranking:** Retrieved documents are reranked based on relevance scores
  using document_reranker.py.
- **Response Generation:** The refined context is used to generate a coherent response
  via OllamaLLM.

 Evaluation:

- **Performance Metrics:** rag_evaluator.py assesses the system's effectiveness using predefined ground truth data.
- **Metrics Calculated:** Recall@5, MAP, and Exact Match scores provide insights into retrieval and response accuracy.

Models and Techniques Integration

- **OllamaLLM** is pivotal in multiple stages:
- **Query Expansion:** Enhances the initial user query for better retrieval.
- **Image Description:** Summarizes images to text for embedding and retrieval.
- **Response Generation:** Crafts answers based on the retrieved and compressed context.
- **Embedding Models** ensure that both text and image data are represented in a vector space conducive to similarity searches:
- **OllamaEmbeddings** and **CLIP** provide versatile embeddings tailored for different data types.
- **ViT-based Embeddings** offer deeper image feature extraction for more nuanced retrieval.
- **Chroma Database** serves as the backbone for storing embeddings, enabling efficient similarity searches essential for the RAG process.
- **Gradio Web Interface** offers an accessible platform for users to interact with the system, making it user-friendly and facilitating easy data ingestion and query testing.
- **Evaluation Metrics** offer a feedback loop, allowing continuous assessment and improvement of the system's retrieval and response capabilities.