TELEGRAM CHATBOT DOCUMENTATION

# Text\_Parsing Package Overview

The Text\_Parsing package is designed for extracting, processing, and chunking text from PDF documents, facilitating semantic analysis and data organization. The package integrates multiple modules, each with specific responsibilities, to streamline the workflow from text extraction to semantic chunking and JSON output.

# Modules

**1.** auxilliaries.py

This module provides utility functions for file handling and JSON operations:

* **make\_directory**: Creates a directory if it doesn't already exist, allowing for organized storage of output files.
* **convert\_final\_map\_to\_json** and **convert\_pages\_map\_to\_json**: These functions convert mapping dictionaries into JSON files for easy storage and access, with filenames derived from the original PDF document.
* **read\_json**: Reads the contents of a JSON file and returns it as a dictionary, enabling the retrieval of previously saved mappings.

**2.** openai\_kit.py

This module interfaces with OpenAI's GPT models to generate text responses based on prompts:

* **get\_gpt\_response**: Sends a prompt to a specified GPT model and retrieves the model's response. This functionality can be used to generate insights or further refine the extracted text.
* **PromptTemplate**: An enumeration that contains predefined prompt templates for guiding the model in tasks such as extracting chapter names from the text, ensuring consistency and clarity in the prompts used.

**3.** semantic\_chunking.py

This module focuses on extracting text from PDFs and dividing it into semantic chunks:

* **SemanticChunking Class**: The central class of this module, responsible for:
  + **extract\_book\_text**: Extracts all text from the specified PDF file.
  + **chunk\_pdf\_content**: Uses a statistical chunking algorithm to divide the extracted text into meaningful segments.
  + **\_gather\_all\_search\_texts**: Gathers subtexts from the chunks for further processing.
  + **\_get\_text\_to\_page\_mapping**: Maps the extracted text back to the original pages of the PDF, aiding in traceability.
  + **\_get\_subtext\_to\_page\_map**: Searches for subtexts in the page text and maps them to their corresponding page numbers.
  + **texts\_to\_page\_mapping**: Creates a comprehensive mapping of text chunks to page numbers, facilitating later analysis.
  + **normalize\_pages**: Ensures that the page numbers in the mapping are sequential and logical.
  + **derive\_final\_maps**: Produces final mappings of text chunks and their corresponding pages for easy reference.

**4.** process\_pdf.py

This module orchestrates the overall process of extracting, chunking, and saving the processed PDF data:

* **Main Workflow**:
  + Defines the PDF file to be processed and sets up folder paths for input and output.
  + Initializes the SemanticChunking class with the path to the PDF.
  + Extracts text from the PDF and chunks it into semantic segments.
  + Creates a mapping of text chunks to their corresponding page numbers.
  + Normalizes the page mappings to ensure logical consistency.
  + Derives final mappings of chunks and their pages.
  + Creates an output directory if it doesn’t exist and saves the mappings to JSON files.

# Workflow

1. **Setup**: Define the PDF file to be processed and establish necessary folder paths for the source PDF and output JSON files.
2. **Text Extraction**: Use the SemanticChunking class from the semantic\_chunking module to extract text from the specified PDF document.
3. **Chunking**: The extracted text is divided into semantic segments using the chunking methods.
4. **Mapping**: Generate mappings that associate text chunks with their respective page numbers in the original PDF.
5. **Normalization**: Ensure the page numbers in the mapping are sequential and logically organized.
6. **Data Storage**: Store the mappings in JSON format for future use or analysis using utility functions from auxilliaries.py.
7. **Model Interaction**: (Optional) Utilize the openai\_kit module to query the GPT model with extracted or chunked data for enhanced insights or text generation.

# Conclusion

The Text\_Parsing package provides a comprehensive toolkit for PDF text extraction and semantic analysis, leveraging advanced chunking techniques and OpenAI's language models. This package is particularly useful for researchers, data analysts, and developers working with educational materials, research papers, or any text-heavy PDF documents.

# Embedding Retrieval from OpenAI

**Package Structure**

**1.** auxiliaries.py

* Purpose: Provides utility functions for managing directories and handling JSON data.
* Key Functions:
  + make\_directory(directory\_name: str): Creates a specified directory if it doesn't already exist.
  + read\_json(json\_folder: str) -> Dict: Reads the first JSON file in a given folder and returns its content.
  + write\_dict\_to\_json(dictionary: dict, folder\_path: str, filename: str): Writes a dictionary to a JSON file at the specified path.

**2.** openai\_kit.py

* Purpose: Interfaces with OpenAI's API for chat completions and embeddings.
* Key Components:
  + get\_chatgpt\_response(instruction: str, openai\_client: OpenAI, temperature: float = 0.0) -> str: Sends an instruction to OpenAI’s API and retrieves the response.
  + **PromptTemplate (Enum)**: Defines templates for structured prompts to be used in API requests.

**3.** prepare\_input.py

* Purpose: Prepares and organizes requests for embedding generation through OpenAI.
* Key Components:
  + create\_request\_number\_to\_chapter\_map(json\_file: Dict) -> Dict: Maps chapter numbers to their names.
  + create\_chapter\_with\_number\_to\_text\_map(json\_file: Dict[str, List[str]]): Creates a mapping of chapter sections to their corresponding text.
  + create\_request\_objects(...): Generates JSON objects for API requests to the embeddings endpoint.
  + create\_input\_files(request\_folder: str) -> List: Uploads JSONL files to OpenAI and returns their file IDs.
  + initialize\_jobs(input\_file\_ids: List) -> List: Starts embedding jobs based on input file IDs.

**4.** retrieve\_embeddings.py

* Purpose: Monitors the status of embedding jobs and retrieves the resulting embeddings.
* Key Functions:
  + \_check\_job\_status(job\_id: str, check\_idx: int) -> Dict: Asynchronously checks the status of a job until completion.
  + monitor\_multiple\_jobs(job\_ids: List): Monitors several batch jobs concurrently, gathering their results.
  + write\_enbeddings\_to\_file(output\_file\_ids: Dict, output\_folder: str): Writes the retrieved embeddings from OpenAI to specified output files.
  + run\_async(func, \*args): Executes an asynchronous function in a blocking manner.

**5.** get\_embeddings.py

* Purpose: Acts as the main script that orchestrates the entire embedding retrieval process.
* Workflow:
  + Reads chapter data from JSON files and prepares mappings.
  + Creates request objects and uploads them to OpenAI.
  + Initializes jobs for processing embeddings.
  + Monitors job completion and retrieves output embeddings.
  + Saves the embeddings to output files in the specified folder.

# Workflow Overview

1. Input Preparation:
   * The process starts in get\_embeddings.py, which reads chapter data from JSON files.
   * It uses the PrepareInputJsonl class to create mappings of chapter sections and generates request objects for embedding.
2. Creating Requests:
   * create\_request\_objects() prepares the embedding requests and stores them in JSONL format.
   * The requests are uploaded to OpenAI using create\_input\_files().
3. Job Initialization:
   * initialize\_jobs() starts the embedding jobs using the uploaded file IDs, queuing them for processing.
4. Monitoring and Retrieval:
   * MonitorAndRetrieveEmbeddings class from retrieve\_embeddings.py monitors the job status asynchronously.
   * Upon job completion, it retrieves the output file IDs and writes the embeddings to specified output files.
5. Output:
   * The final embeddings are saved in the designated output folder, ready for further analysis or use.

# Chatbot\_Module Overview

The Chatbot\_Module package is designed to create an interactive chatbot that retrieves information from user-provided materials and generates responses using OpenAI's API and Pinecone's vector database. It integrates various components, including embedding retrieval, query processing, and communication via a Telegram bot.

# Modules

**1.** auxiliaries.py

This module provides utility functions and type definitions used throughout the package.

Main Functions**:**

* Type definitions for lists, dictionaries, and other data structures.

**2.** embedding\_matrix\_loader.py

This module handles loading vectorizers, converting queries to vector representations, and hybrid scaling of dense and sparse embeddings.

Main Functions**:**

* **load\_vectorizer**: Loads a TF-IDF vectorizer from a specified directory.
* **convert\_string\_query\_to\_vectors**: Converts a user query into dense and sparse vector representations.
* **sparse\_matrix\_to\_dict**: Converts a sparse matrix into a dictionary format.
* **hybrid\_scale**: Combines dense and sparse vectors using a convex combination based on a specified alpha value.

**3.** openai\_kit.py

This module serves as a wrapper around OpenAI's API functionalities, enabling the retrieval of embeddings and responses from the ChatGPT model.

Main Functions**:**

* **get\_embedding**: Takes a text input and retrieves its embedding from OpenAI's API.
* **get\_chatgpt\_response**: Sends a user instruction to the ChatGPT model and returns the model's response.

Enums**:**

* **PromptTemplate**: Defines various prompt templates used in the interaction with the chatbot.

**4.** pinecone\_kit.py

This module facilitates interaction with the Pinecone vector database, allowing for the querying and retrieval of relevant documents.

Main Functions**:**

* **get\_pinecone\_index\_client**: Retrieves a Pinecone index client for a specified index name.
* **get\_pinecone\_query\_result**: Queries Pinecone for results based on dense and sparse vector inputs.
* **parse\_texts\_from\_pinecone**: Parses the retrieved results from Pinecone to extract relevant texts and metadata.

**5.** query\_processor.py

This module processes user queries by interacting with the OpenAI API and Pinecone embedding service. It takes user input, converts it into vector representations, queries Pinecone for relevant text, and generates a response using GPT-4.

Main Functions**:**

* **get\_gpt\_response\_from\_pinecone\_text**: Generates a response from GPT-4 based on a single piece of text retrieved from Pinecone and the user's query.
* **get\_gpt\_response\_from\_pinecone\_texts**: Processes multiple texts from Pinecone, generating responses and appending page information when relevant.
* **process\_query**: Takes a user query, retrieves relevant texts from Pinecone, and generates an appropriate response using OpenAI's API.

Enums**:**

* **AlphaValues**: Enum defining possible alpha values for hybrid scaling in query processing.

**6.** activate\_telegram\_bot.py

This module sets up a Telegram bot that interacts with users, sending and receiving messages.

Main Functions**:**

* **start**: A command handler that greets the user when they use the /start command.
* **bot\_response**: A message handler that responds to user messages with answers generated by the chatbot.

# How It All Works Together

1. **User Interaction**: Users interact with the chatbot via the Telegram bot implemented in activate\_telegram\_bot.py.
2. **Query Processing**: When a user sends a message, bot\_response in the Telegram bot processes the input using the process\_query function in query\_processor.py.
3. **Embedding and Retrieval**: The user's query is converted into vector representations using the vectorizer in embedding\_matrix\_loader.py, and relevant texts are retrieved from Pinecone using functions in pinecone\_kit.py.
4. **Response Generation**: The retrieved texts are used to generate responses from OpenAI's API, utilizing the prompt templates defined in openai\_kit.py.
5. **Response Delivery**: The generated response is sent back to the user through the Telegram bot.