CODE DOCUMENTATION

Class Initialization

The HierarchicalDocumentSegmenter class is designed to process and segment a PDF document into structured hierarchical segments based on formatting and heading rules. When initialized with a pdf_path, and optionally an LLM API key and provider, it sets up various attributes like full_text, formatted_blocks, and segments. These are used throughout the pipeline to store extracted content, formatting metadata, and structured segment information.

extract_text_with_formatting

This function reads the PDF using the PyMuPDF (fitz) library and extracts text along with rich formatting details. It traverses through each page and captures blocks, lines, and spans, preserving font details (like size and type), position, and page number. It stores all this information in the formatted_blocks list and compiles the plain text into the full_text string. This function returns the raw full text of the PDF while keeping the structural details required for downstream segmentation tasks.

identify_candidate_headings_rule_based

This function identifies potential section headings using two main strategies: font size differentiation and pattern matching. It groups text spans by font properties and assumes the most frequent font group to represent the body text. Any text span significantly larger than the body text is considered a heading candidate. Additionally, it uses regex patterns to match typical section heading formats (e.g., "1.", "Chapter 2:", "Section 3.1"). The output is a list of candidate headings, each with metadata like text, size, and position.

assign_levels_by_rules

This function assigns hierarchical levels (e.g., section, subsection) to the identified candidate headings based on their font sizes. Font sizes are sorted in descending order, and higher-level headings are those with larger sizes. The function assigns a level (from 1 to 3) to each heading and also extracts a cleaned version of the heading by removing numeric prefixes using regex. The result is a sorted list of verified_headings that will be used to segment the full text.

create_hierarchical_segments

This function constructs logical text segments using the verified headings. For each heading, it identifies the text range up to the next heading and extracts the

corresponding content. It cleans the text (removing special characters, newlines) and attaches metadata like heading level, segment title, and position in the document. It also invokes helper functions to extract additional metadata such as dates and sources (e.g., author names) from the segment text. The output is a list of segments stored in self.segments.

extract_segment_date

This utility function attempts to identify a date within a given text segment using common date formats such as YYYY-MM-DD, MM/DD/YYYY, and Month Day, Year. It searches the segment using regex and returns the first match if found. If no date is found, it returns None.

extract_segment_source

This function tries to determine the source or author of a segment. It first searches for explicit attribution patterns like "By John Doe" or "Source: XYZ". If not found, it uses the Stanza NLP pipeline to extract named entities of type "PERSON" within the first few characters of the text. These are potential author names. Optionally, it sends this information to an LLM-based verifier (VerifyWithLLM) to confirm the author's relevance. The result is either a valid author name or None.

clean_text

This helper function performs basic text cleaning by replacing newlines, special characters like – or , and trimming whitespaces. This ensures cleaner output for display or further processing.

extract_named_entities

This function uses the Stanza NLP library to extract named entities such as persons, organizations, locations, and dates from each segment. It processes each segment individually and collects entities using the built-in Stanza NER model. Duplicate entities are removed before attaching them to the corresponding segment. The output is an enriched list of segments with entity metadata.

process_document

This is the main driver function of the class. It sequentially executes the pipeline: extracting text and formatting, identifying candidate headings, assigning levels, creating hierarchical segments, and extracting named entities. It returns the final list of enriched segments ready for export or analysis.

export_to_json

This utility function writes the final list of hierarchical segments, along with their metadata and extracted entities, into a JSON file at the specified output_path. It ensures UTF-8 encoding and readable formatting with indentation. The path to the saved JSON file is returned.

_verify_author_with_llm

Function inside the VerifyWithLLM class is designed to check whether a detected person's name is actually the author or contributor of a document segment. It takes two inputs: candidate_author, which is the name to verify, and context, which is a piece of text around where the name appeared. The function builds a prompt asking the LLM directly whether the candidate is the author, requesting a simple "YES" or "NO" answer to avoid confusion. It then sends this prompt to Groq's LLaMA 3 model using the Groq API, setting parameters like low temperature to keep the output focused. Once it gets a response, the function cleans it up, checks if the answer contains "YES", and if so, returns the candidate's name. Otherwise, it returns None. In case the API call fails for any reason (like a bad API key or a network issue), the function handles the error gracefully and simply returns the original candidate name instead of breaking the flow.

main.py file

The main.py script loads a PDF file, processes it to extract and segment its content hierarchically, and saves the result as a JSON file. It begins by loading environment variables (like the Groq API key) using dotenv, then defines a process_pdf function that initializes the HierarchicalDocumentSegmenter, runs the full processing pipeline—including text extraction, heading detection, segmentation, and named entity recognition—and exports the structured segments to a JSON file. When run directly, it processes a specific PDF (Gen AI assignment.pdf) and outputs the results to segments.json.