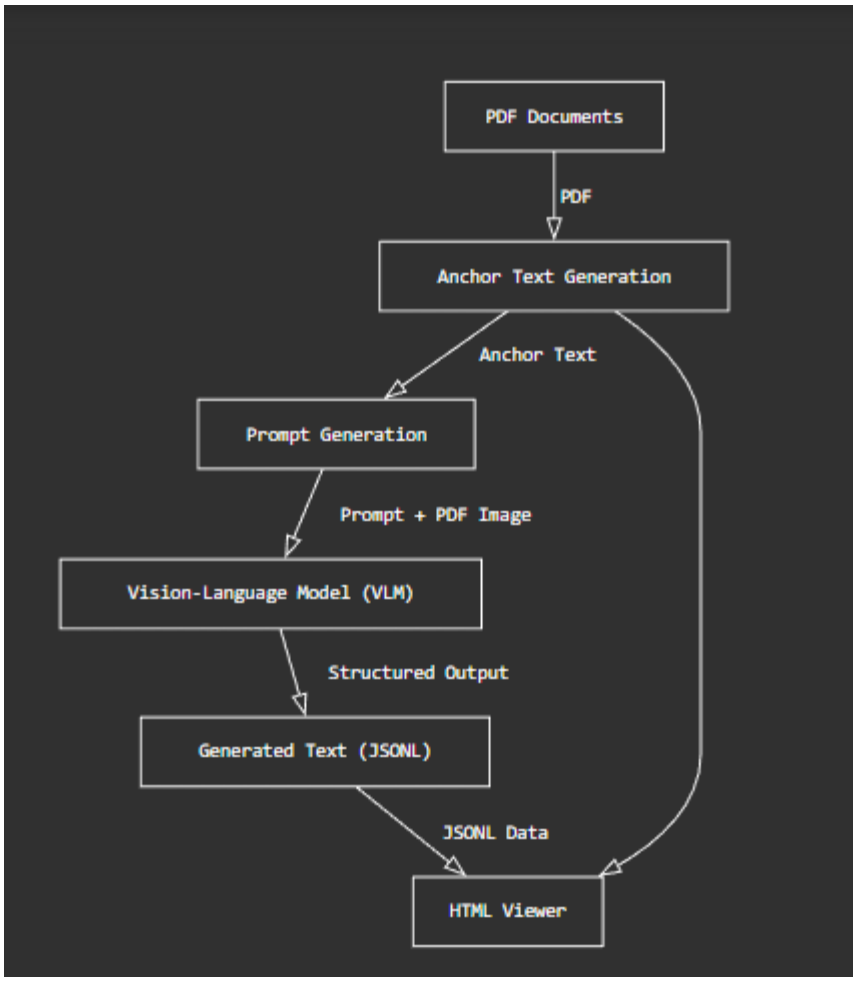


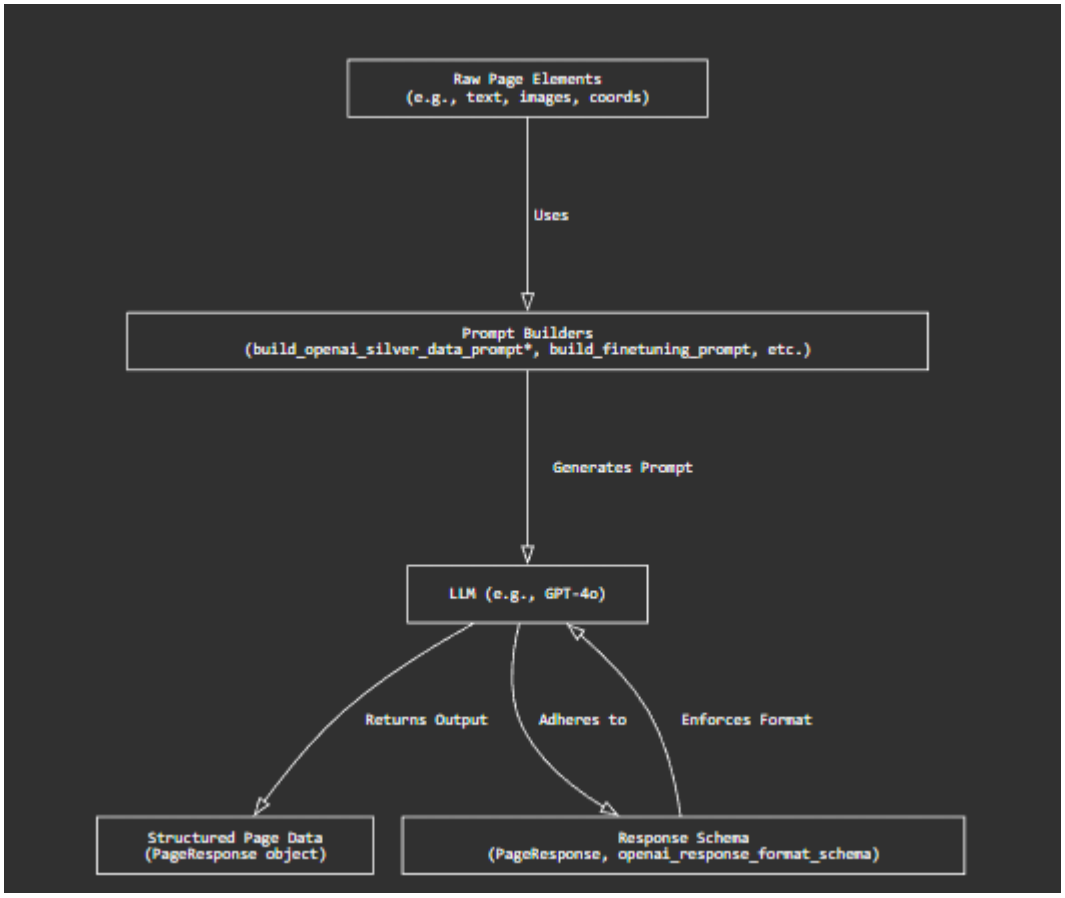
Prompt generation for Optical Character Recognition (OCR) tasks within **olmOCR** facilitates structured communication with large language models (LLMs) for document processing, encompassing text, tables, and equation extraction, and defining expected response formats. This framework allows for the construction of various prompts, including those for silver data generation, finetuning, and YAML-based instructions, which guide LLMs to produce specific outputs. A key component is the enforcement of structured output schemas for LLM responses, ensuring predictable and parsable results. This is achieved through the definition of dataclasses like **PageResponse** and the generation of corresponding JSON schemas in **olmocr/prompts/prompts.py**.

A related capability is the generation of "anchor text" from PDF documents to enhance Visual Language Model (VLM) understanding. This process, detailed in **olmocr/prompts/anchor.py**, involves utilizing multiple PDF parsing engines to extract text and structure page reports into textual descriptions that include coordinate

information. This approach aims to provide VLMs with a more contextually rich representation of document content beyond simple text extraction.

For reviewing the outputs of these processes, **olmOCR** includes functionality for generating interactive HTML previews from JSONL files, as demonstrated in **olmocr/viewer/dolmaviewer.py**. These previews visualize document content, including text snippets, rendered PDF pages as base64 WebP images, and associated metadata. The system supports both local and S3 paths for input JSONL and source PDF files, allowing for flexible data access. The HTML viewer, leveraging templates such as **olmocr/viewer/dolmaviewer_template.html** and **olmocr/viewer/dolmaviewer_merged_template.html**, offers toggleable views for rendered Markdown (with LaTeX support via KaTeX) and raw text, as well as navigation features for merged documents. This interactive visualization tool is crucial for human verification and annotation of OCR results. See Web-Based Review and Annotation Tools for more information on the review process. The comprehensive KaTeX rendering and comparison functionality, including server-side rendering with Playwright and MathML extraction, is further elaborated in KaTeX Rendering and Equation Comparison.

LLM Prompt Construction and Schema Enforcement



The olmocr project constructs various types of prompts for large language models (LLMs) to facilitate structured communication during document processing tasks. This includes generating prompts for creating "silver data," finetuning models, and producing YAML-based outputs. A core aspect of this process involves defining and enforcing structured output schemas for LLM responses using dataclasses and JSON schema.

Prompt construction is handled by functions such as those found in olmocr/prompts/prompts.py. These functions are designed to instruct LLMs on how to process different elements within a PDF document page, such as converting equations to LaTeX, formatting tables into Markdown or HTML, and handling specific document features like headers, footers, handwriting, and figure labels. For instance, prompts like build_openai_silver_data_prompt_v3_simple provide detailed instructions, including page dimensions, and enforce specific formatting rules for LaTeX and HTML elements.

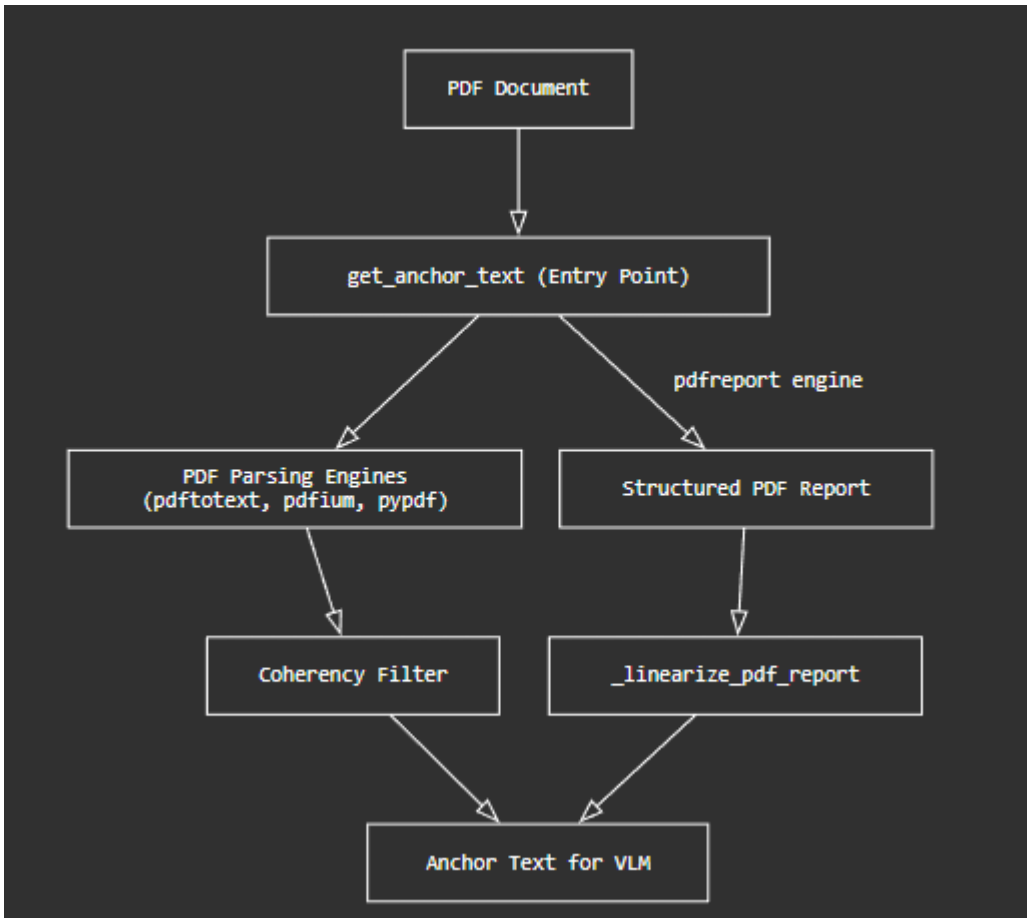
To ensure that LLMs return responses in a predictable and parseable format, the project utilizes a PageResponse dataclass. This dataclass, defined in olmocr/prompts/prompts.py, specifies the expected structure of an LLM's output for a single page, including fields for language identification, rotation validation, and content type flags (e.g., is_table, is_diagram). The __post_init__ method within PageResponse includes validation logic to ensure that fields like rotation_correction adhere to predefined values.

Furthermore, a JSON schema is generated for the PageResponse dataclass via the openai_response_format_schema function in olmocr/prompts/prompts.py. This schema serves as a formal contract, guiding LLMs to produce output that conforms to the defined structure. This strict schema enforcement is crucial for robust downstream processing and analysis of LLM-generated content.

Simpler prompts, such as those generated by build_finetuning_prompt, are also available for fine-tuning pre-trained models, focusing primarily on extracting natural text without extensive formatting instructions. For outputs that require embedded metadata, functions like build_no_anchoring_yaml_prompt and build_no_anchoring_v4_yaml_prompt generate prompts that instruct LLMs to produce Markdown with YAML front matter, including page metadata like language and content type. The v4 version specifically requests HTML for tables and includes figure labeling instructions. The extract_raw_text utility aids in

retrieving specific content blocks from these prompt strings using defined start and end tags.

PDF Anchor Text Generation for Visual Language Models



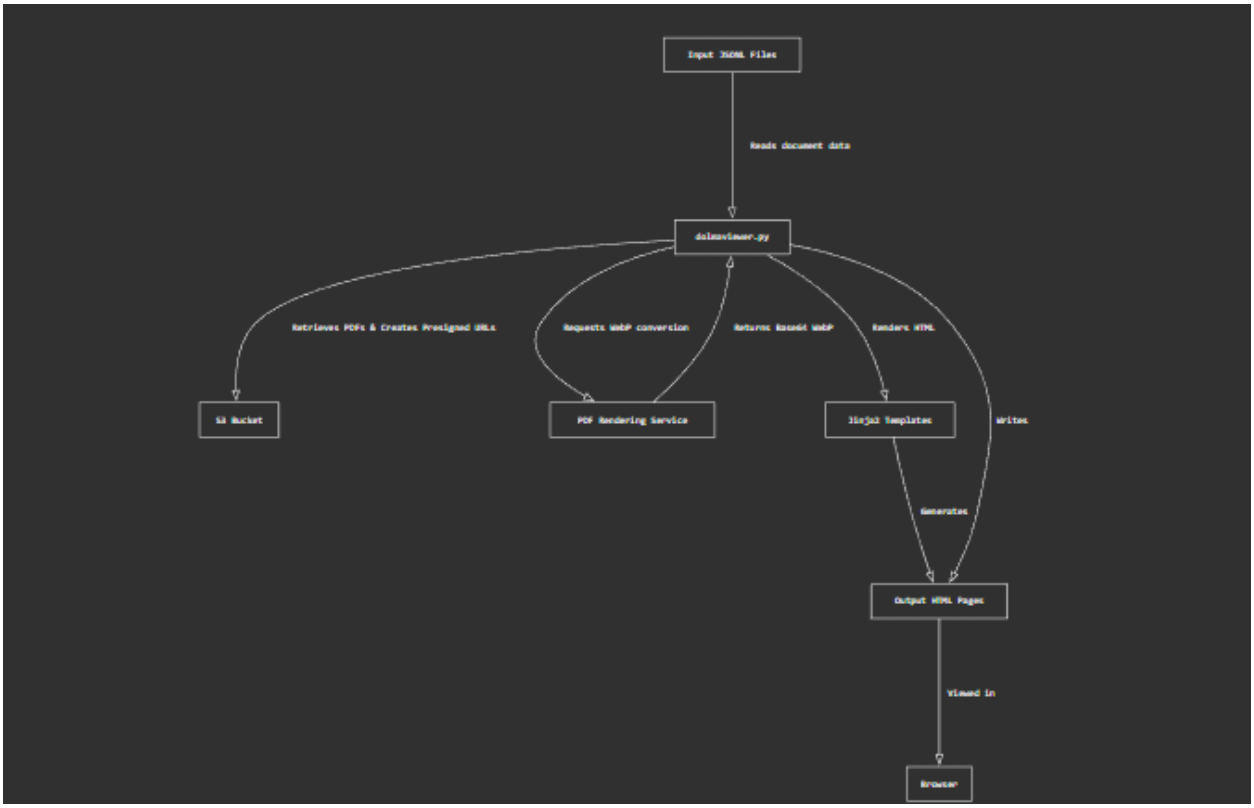
The olmocr project generates "anchor text" from PDF documents to improve the comprehension capabilities of Visual Language Models (VLMs). This process involves extracting and structuring text in a way that provides spatial and contextual information about the document content. The primary function for this is get_anchor_text, found in olmocr/prompts/anchor.py, which supports multiple PDF parsing engines to extract text from a specified page.

Several PDF parsing engines can be employed: pdftotext, pdfium, and pypdf offer different methods for extracting raw text. Additionally, a "topcoherency" engine selects

the most coherent text output by comparing results from these different methods. A more advanced **"pdfreport"** engine first creates a structured **PageReport** of text and image elements using **pypdf**'s internal mechanisms, capturing their positions and relationships. This report is then linearized into a textual description that includes coordinate information for each element, providing VLMs with a visually-aware representation of the page content. This linearization process, handled by **_linearize_pdf_report** in **olmocr/prompts/anchor.py**, aims to preserve critical layout information by prioritizing elements at the visual "edges" of the page and merging overlapping image elements to create a more accurate spatial understanding.

The system also includes utilities for robust text cleaning and standardization, such as fixing Unicode issues and truncating long text elements while preserving context. The overall goal is to create rich, structured textual anchors that help VLMs better understand the layout and content of PDF documents. For a more general overview of prompt construction and schema enforcement for LLMs, see LLM Prompt Construction and Schema Enforcement.

HTML Document Viewer for JSONL Outputs



The **olmocr** project includes functionality for generating interactive HTML previews from JSONL-formatted document data. This allows for visualizing processed document content and metadata in a user-friendly web interface. These previews can be generated for individual documents or as merged views encompassing multiple documents.

The core of this functionality lies within the **olmocr/viewer** directory, specifically orchestrated by the **olmocr/viewer/dolmaviewer.py** script. This script processes JSONL input, which can be sourced from local files or Amazon S3 buckets, and converts specified PDF pages into base64-encoded WebP images for embedding in the HTML output. For PDFs stored in S3, the viewer generates time-limited presigned URLs to ensure direct access to the source documents from the HTML.

The generated HTML previews, based on templates such as **olmocr/viewer/dolmaviewer_template.html** for individual documents and **olmocr/viewer/dolmaviewer_merged_template.html** for merged views, offer several features:

- **Content Display:** Document content is organized into logical sections, displaying text alongside corresponding images.
- **Metadata Presentation:** Comprehensive metadata, including source information, processing versions, creation dates, page counts, and language details, is presented in a structured format.
- **Toggleable Views:** Users can switch between a "Markdown View" and a "Raw Text View." The Markdown view leverages client-side JavaScript libraries like **marked.js** for Markdown rendering and **KaTeX** for typesetting LaTeX mathematical expressions. This client-side rendering allows for dynamic content updates without requiring a page reload.
- **Navigation:** For merged document views, a dropdown menu facilitates quick navigation between different documents.

The system is designed to handle document data efficiently, utilizing concurrent processing for PDF rendering and HTML generation, especially when dealing with numerous documents. This approach allows for rapid visualization and review of OCR outputs.