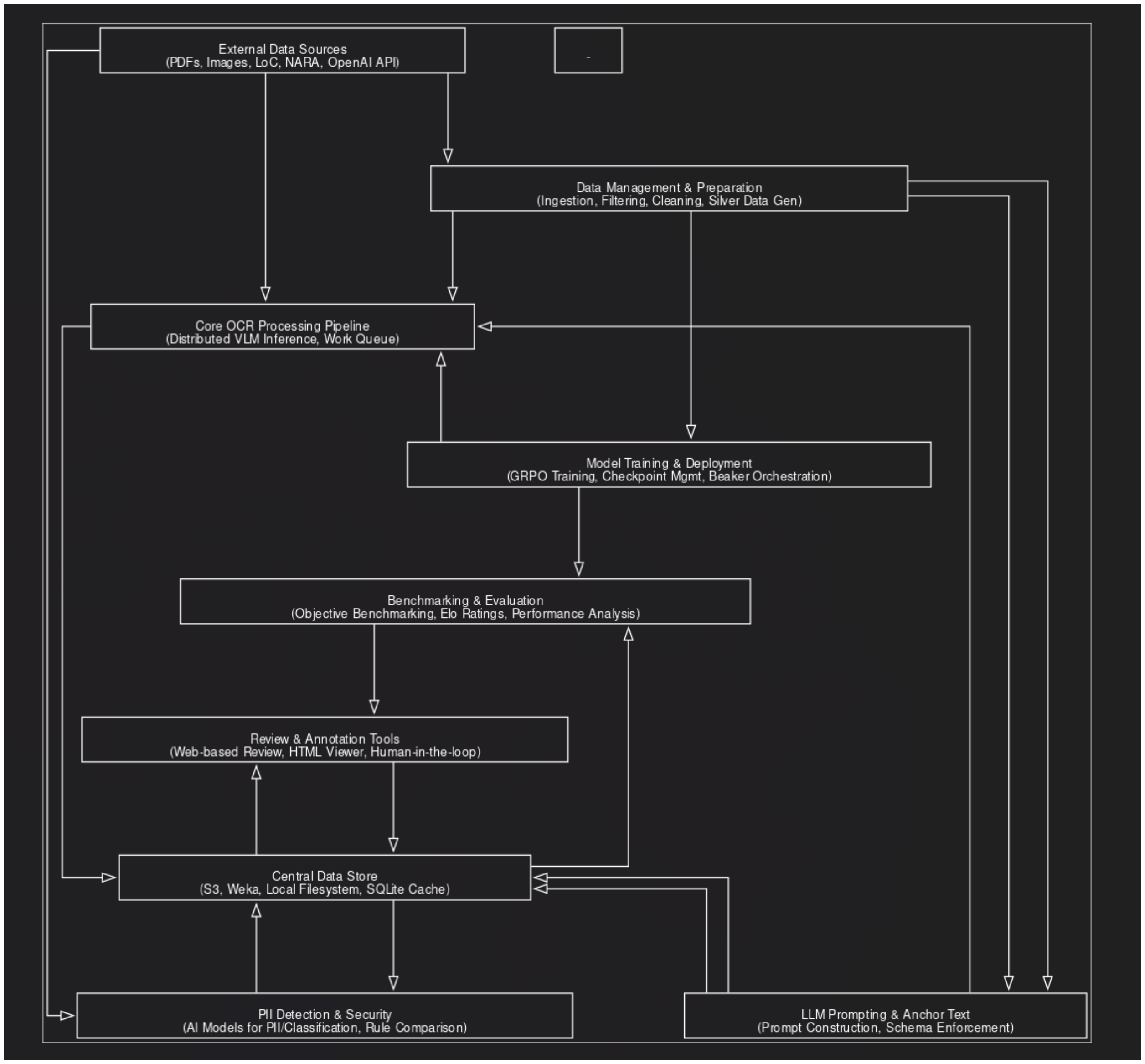

External Data Sources
(PDFs, Images, LoC, NARA, OpenAI API)
-
Data Management & Preparation
(Ingestion, Filtering, Cleaning, Silver Data Gen)
Core OCR Processing Pipeline
(Distributed VLM Inference, Work Queue)
Model Training & Deployment
(GRPO Training, Checkpoint Mgmt, Beaker Orchestration)
Benchmarking & Evaluation
(Objective Benchmarking, Elo Ratings, Performance Analysis)
Review & Annotation Tools
(Web-based Review, HTML Viewer, Human-in-the-loop)
Central Data Store
(S3, Weka, Local Filesystem, SQLite Cache)
PII Detection & Security
(AI Models for PII/Classification, Rule Comparison)
LLM Prompting & Anchor Text
(Prompt Construction, Schema Enforcement)

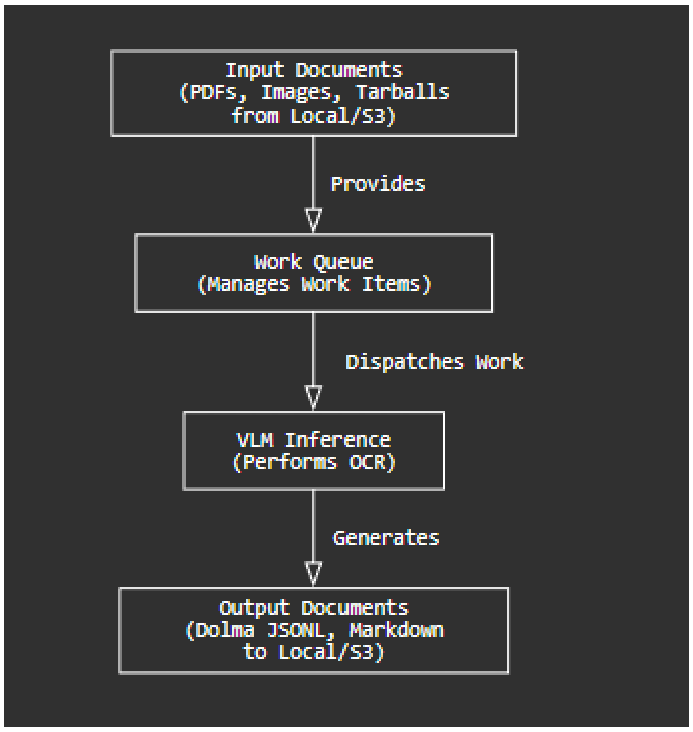
The olmocr project provides a framework for document-level Optical Character Recognition (OCR). It converts image-based documents, such as PDFs, into structured text formats for analysis and further processing. The system offers capabilities for objective evaluation of OCR tools, distributed document processing, and structured data output across diverse document types.

The framework integrates several key components:

- **OCR Conversion Pipeline:** This component orchestrates the OCR conversion process for single or multiple documents. It manages work queues and coordinates

Vision Language Model (VLM) inference to generate structured Dolma documents or Markdown outputs.

Core OCR Pipeline and Architecture


Input Documents (PDFs, Images, Tarballs from Local/S3)
Provides
Work Queue (Manages Work Items)
Dispatches Work
VLM Inference (Performs OCR)
Generates
Output Documents (Dolma JSONL, Markdown to Local/S3)

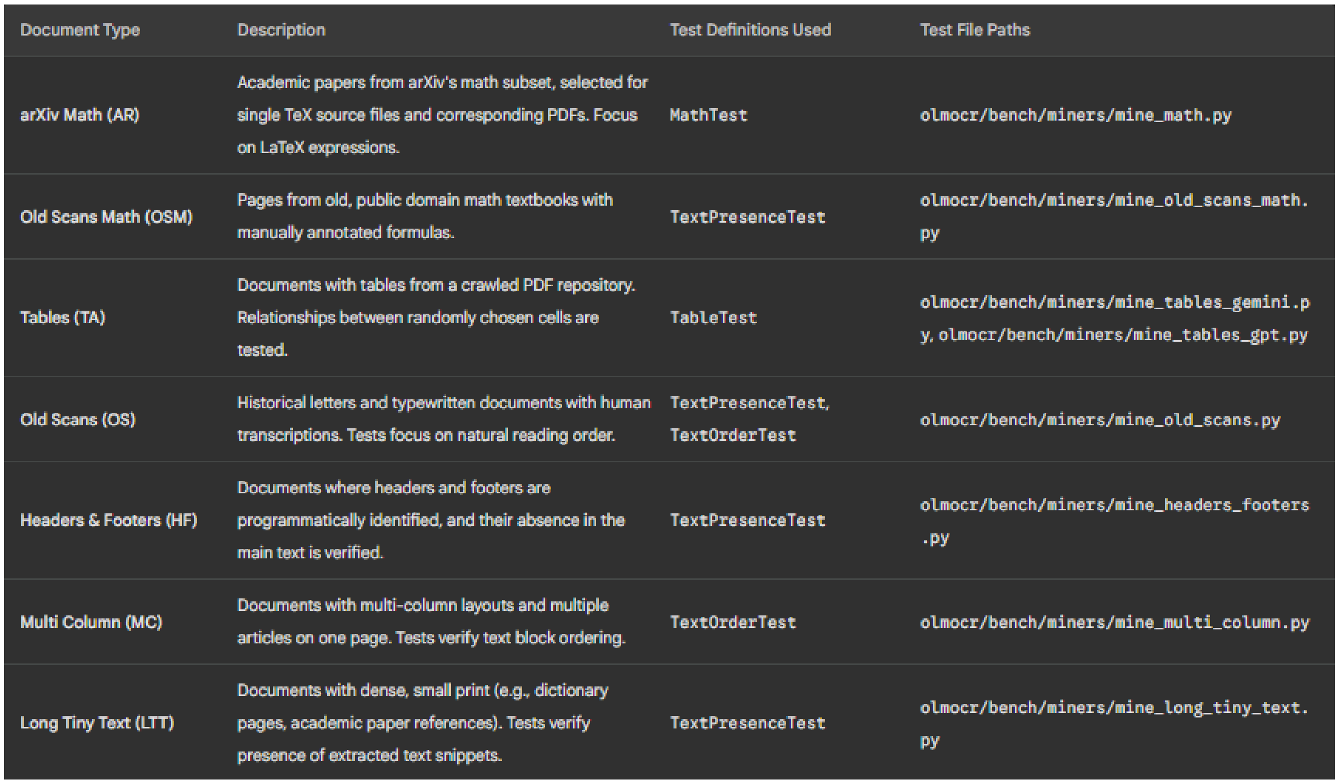
The **olmocr** project provides a framework for orchestrating the conversion of single or multiple PDF documents and images into structured formats. This process leverages a distributed architecture, managing work queues and coordinating Vision Language Model (VLM) inference to produce outputs such as Dolma documents (JSONL) or Markdown.

At its core, the conversion process is managed by a central pipeline that parses command-line arguments, sets up logging, and executes tasks asynchronously. This pipeline can handle documents from local storage or cloud-based object storage like S3. A key component of this architecture is the work queue, which is designed to be fault-tolerant and asynchronous. It manages individual units of work, each representing a document or a group of documents to be processed. This work queue uses pluggable storage backends, meaning it can operate with a local filesystem or integrate with S3 for managing its state and tracking completed tasks. This modularity allows the system to scale and adapt to different deployment environments.

For each page within a document, the system builds structured prompts that include base64-encoded images of the page. These prompts are then sent to a VLM for inference. The system incorporates mechanisms for handling image rotation, retrying failed requests, and applying exponential backoff to ensure robust communication with the VLM. The responses from the VLM are parsed to extract structured data, which includes text content and metadata.

The output from the VLM inference, comprising page-level results, is aggregated and transformed into standardized formats. These include "Dolma" JSON documents, which are suitable for downstream data ingestion and analysis, and human-readable Markdown files. This dual output capability supports both machine-based processing and human review of the extracted content. The overall process is designed to track various metrics, such as token usage, processed pages, and retries, and to monitor the status of worker processes, ensuring operational visibility and performance tracking.

Objective Benchmarking Framework and Document Types



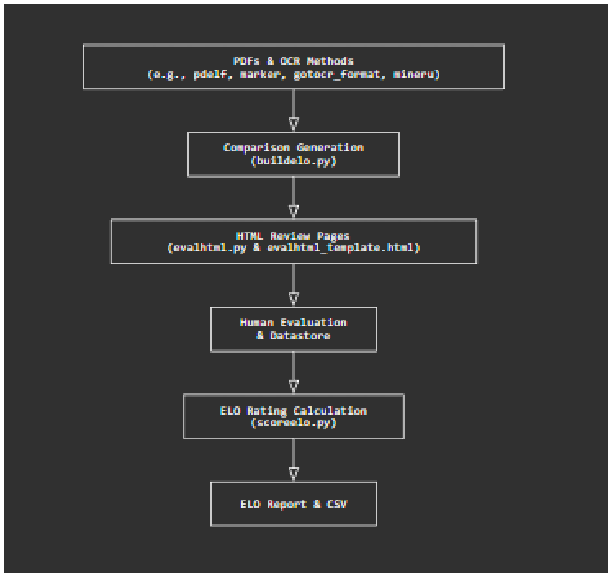
Document Type	Description	Test Definitions Used	Test File Paths
arXiv Math (AR)	Academic papers from arXiv's math subset, selected for single TeX source files and corresponding PDFs. Focus on LaTeX expressions.	`MathTest`	`olmocr/bench/miners/mine_math.py`
Old Scans Math (OSM)	Pages from old, public domain math textbooks with manually annotated formulas.	`TextPresenceTest`	`olmocr/bench/miners/mine_old_scans_math.py`
Tables (TA)	Documents with tables from a crawled PDF repository. Relationships between randomly chosen cells are tested.	`TableTest`	`olmocr/bench/miners/mine_tables_gemini.py`, `olmocr/bench/miners/mine_tables_gpt.py`
Old Scans (OS)	Historical letters and typewritten documents with human transcriptions. Tests focus on natural reading order.	`TextPresenceTest`, `TextOrderTest`	`olmocr/bench/miners/mine_old_scans.py`
Headers & Footers (HF)	Documents where headers and footers are programmatically identified, and their absence in the main text is verified.	`TextPresenceTest`	`olmocr/bench/miners/mine_headers_footers.py`
Multi Column (MC)	Documents with multi-column layouts and multiple articles on one page. Tests verify text block ordering.	`TextOrderTest`	`olmocr/bench/miners/mine_multi_column.py`
Long Tiny Text (LTT)	Documents with dense, small print (e.g., dictionary pages, academic paper references). Tests verify presence of extracted text snippets.	`TextPresenceTest`	`olmocr/bench/miners/mine_long_tiny_text.py`

The **olmocr** project includes an objective framework for evaluating Optical Character Recognition (OCR) systems, named **olmocr-bench**. This framework evaluates OCR performance against machine-checkable "facts" rather than subjective metrics, leveraging a diverse set of seven document types. These types are chosen to represent distinct challenges for OCR technology: arXiv Math (AR), Old Scans Math (OSM), Tables (TA), Old Scans (OS), Headers Footers (HF), Multi Column (MC), and Long Tiny Text (LTT).

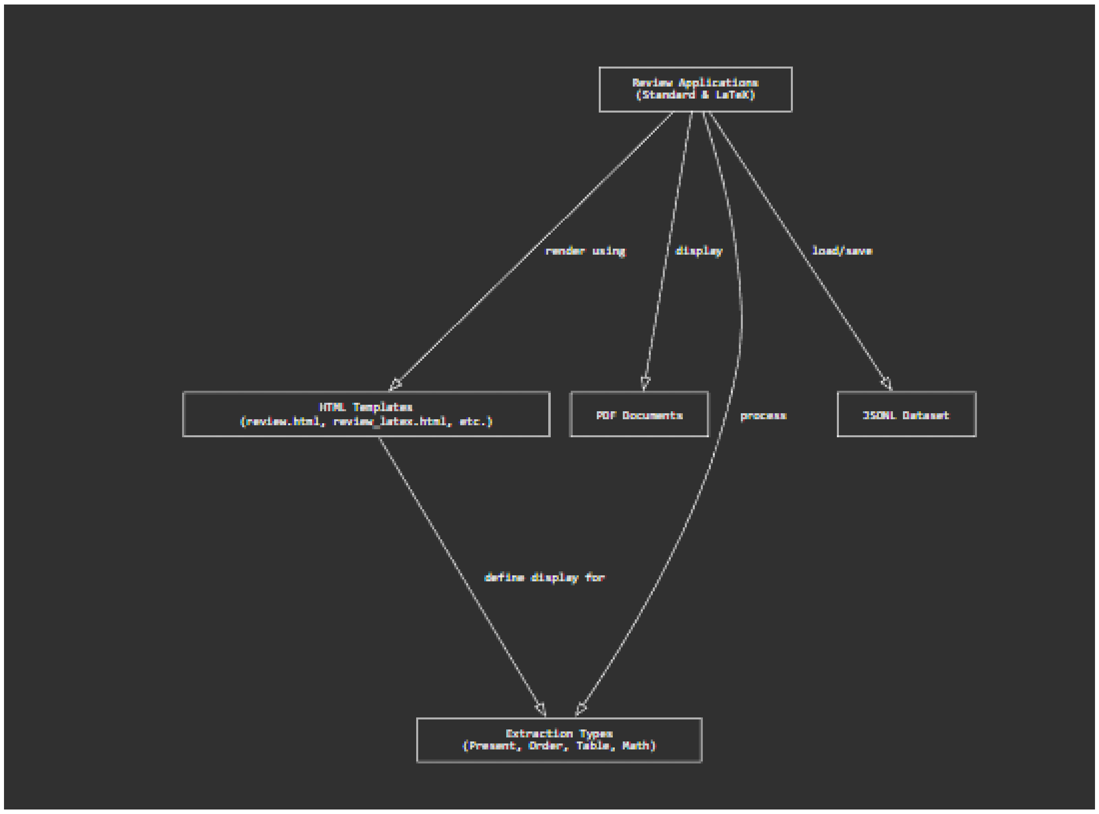
The evaluation process is managed by a script within **olmocr/bench/benchmark.py**, which orchestrates the loading of test cases from JSONL files and the concurrent evaluation of various OCR tools. It calculates overall scores and employs bootstrap confidence intervals to ensure statistical robustness. The results can be presented in a detailed HTML report through **generate_html_report**. A key design principle of the benchmark is to calculate the overall score as an average of scores derived from individual JSONL files, thereby ensuring that each test suite contributes equally to the final assessment.

Specific test cases for each document type are defined within **olmocr/bench/tests.py**. These tests validate different aspects of OCR output. For instance, text presence and order are verified by **TextPresenceTest** and **TextOrderTest**, while structural accuracy for tables is assessed by **TableTest**. The correctness of mathematical equations is evaluated using **MathTest**. For **TableTest**, functions like **parse_markdown_tables** and **parse_html_tables** are used to handle various table formats, and for **MathTest**, **render_equation** and **compare_rendered_equations** are employed to compare mathematical expressions. Fuzzy matching techniques are extensively utilized across these tests to account for minor variations. For details on tools used for human verification and correction of test results, refer to

Human-in-the-Loop Evaluation and Review Page Generation



Web-Based Review and Annotation Tools


Review Applications (Standard & LaTeX)
render using
display
load/save
HTML Templates (review.html, review_latex.html, etc.)
PDF Documents
process
JSONL Dataset
define display for
Extraction Types (Present, Order, Table, Math)

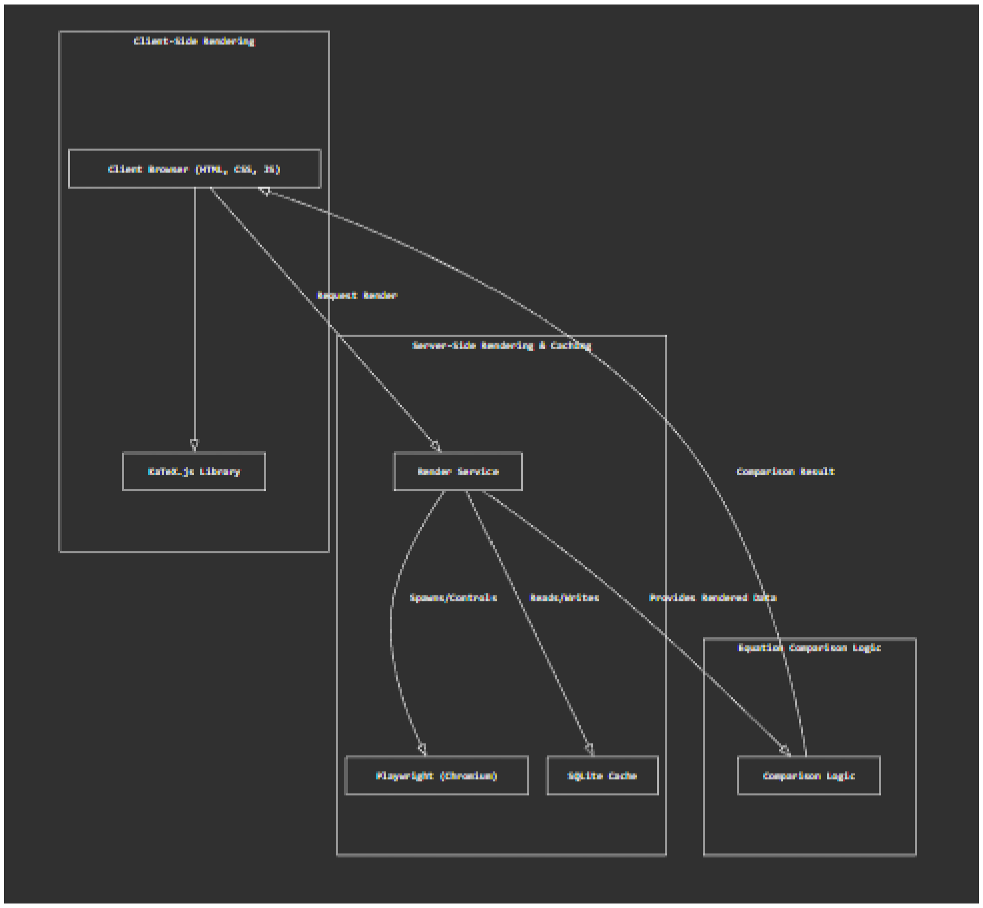
The **olmocr** project includes web-based applications designed to facilitate human verification, correction, and annotation of OCR test results. These tools support an interactive workflow for reviewing various extraction types and specialized support for mathematical equations.

The general review application, implemented in **olmocr/bench/review_app.py**, provides an interface for evaluating test results associated with PDF documents. Users can navigate through a collection of PDFs, examine individual tests (such as text presence, tables, or natural reading order), and mark them as "verified" or "rejected." The application tracks completion statistics and atomically saves changes to a JSONL dataset. This ensures that the progress is persistently recorded without data corruption. Navigation features allow users to move between PDFs, prioritizing those with unchecked tests, or to cycle through all documents in a "force" mode.

For LaTeX equations, a specialized review application is provided in **olmocr/bench/review_app_latex.py**. This tool is tailored to the unique challenges of mathematical content. It leverages client-side technologies such as **pdfjsLib** for rendering

PDFs within the browser and MathJax for accurately typesetting LaTeX equations. Reviewers can view extracted equations alongside the PDF, verify their correctness, reject erroneous extractions, or directly edit the raw LaTeX source. The application supports similar navigation, status tracking, and atomic saving mechanisms as the general review tool, allowing for efficient and precise annotation of mathematical content. Both review applications utilize HTML templates located in olmocr/bench/templates to render their user interfaces, providing a consistent and interactive experience.

KaTeX Rendering and Equation Comparison



The framework provides a robust system for rendering and comparing mathematical equations written in LaTeX, a critical component for evaluating OCR systems on scientific and technical documents. This functionality involves both client-side and server-side processing to ensure accurate display and rigorous comparison of mathematical expressions.

On the client side, olmocr/bench/katex/auto-render.min.js and olmocr/bench/katex/katex.min.js are JavaScript libraries that enable the automatic detection and rendering of LaTeX math within a web page. These scripts scan a given HTML element for mathematical expressions, identified by configurable delimiters (e.g., \$\$...\$\$, \\(...\\)), and convert them into visually formatted HTML using the core KaTeX functionality. This allows for interactive display and review of equations within web-based applications, such as the specialized review tool for LaTeX equations found in olmocr/bench/review_app_latex.py. Styling for these rendered equations is provided by olmocr/bench/katex/katex.min.css, which includes extensive font definitions and CSS rules to accurately represent various mathematical symbols and structures.

For more detailed analysis and comparison, server-side rendering is orchestrated by olmocr/bench/katex/render.py. This module uses Playwright to launch a headless Chromium browser instance, effectively creating a controlled environment for rendering LaTeX strings with KaTeX. During this process, it extracts not only the visual rendering but also critical metadata, including the bounding boxes of individual text span elements and the MathML representation of the equation. MathML provides a structural and semantic representation of mathematical content, which is crucial for objective comparison. To optimize performance and avoid redundant computations, rendered equations and their associated metadata are cached in a SQLite database, indexed by a SHA1 hash of the LaTeX input.

The core of the equation comparison logic resides in the compare_rendered_equations function within olmocr/bench/katex/render.py. This function takes two rendered equation objects and performs a multi-stage comparison. Initially, it normalizes and compares their MathML representations to check for semantic equivalence. If the MathML differs, a more granular, span-based comparison is initiated. This involves analyzing the bounding boxes and text content of individual elements within the rendered equations, identifying neighboring elements, and attempting to establish a mapping between corresponding spans in both equations. This intricate process allows the framework to assess the accuracy of OCR systems in transcribing not just the text of an equation, but also its structural integrity and visual layout, supporting various LaTeX commands and semantic equivalences. This forms a key part of the objective benchmarking framework detailed in Objective Benchmarking Framework and Document Types.