

The **olmOCR** framework provides capabilities for detecting Personally Identifiable Information (PII) within Dolma OCR documents. This functionality leverages vision-based AI models, such as ChatGPT, to classify document types and identify various forms of PII. The system is designed to generate structured attributes and reports from this analysis, processing diverse compressed JSONL files stored locally or in S3.

The process typically involves scanning OCR'd document pages, often in PDF format, to identify PII. This is achieved by rendering PDF pages as images and submitting them to a vision model with a detailed prompt for PII identification. The model's response is then parsed into a structured schema, ensuring consistent and validated output. This structured data categorizes PII annotations and generates summary reports. For example, **PIIAnnotation** in [scripts/pii/autoscan\\_dolmadocs.py](#) is a Pydantic model that defines the structured schema for PII annotations returned by ChatGPT, including various boolean flags for specific PII types.

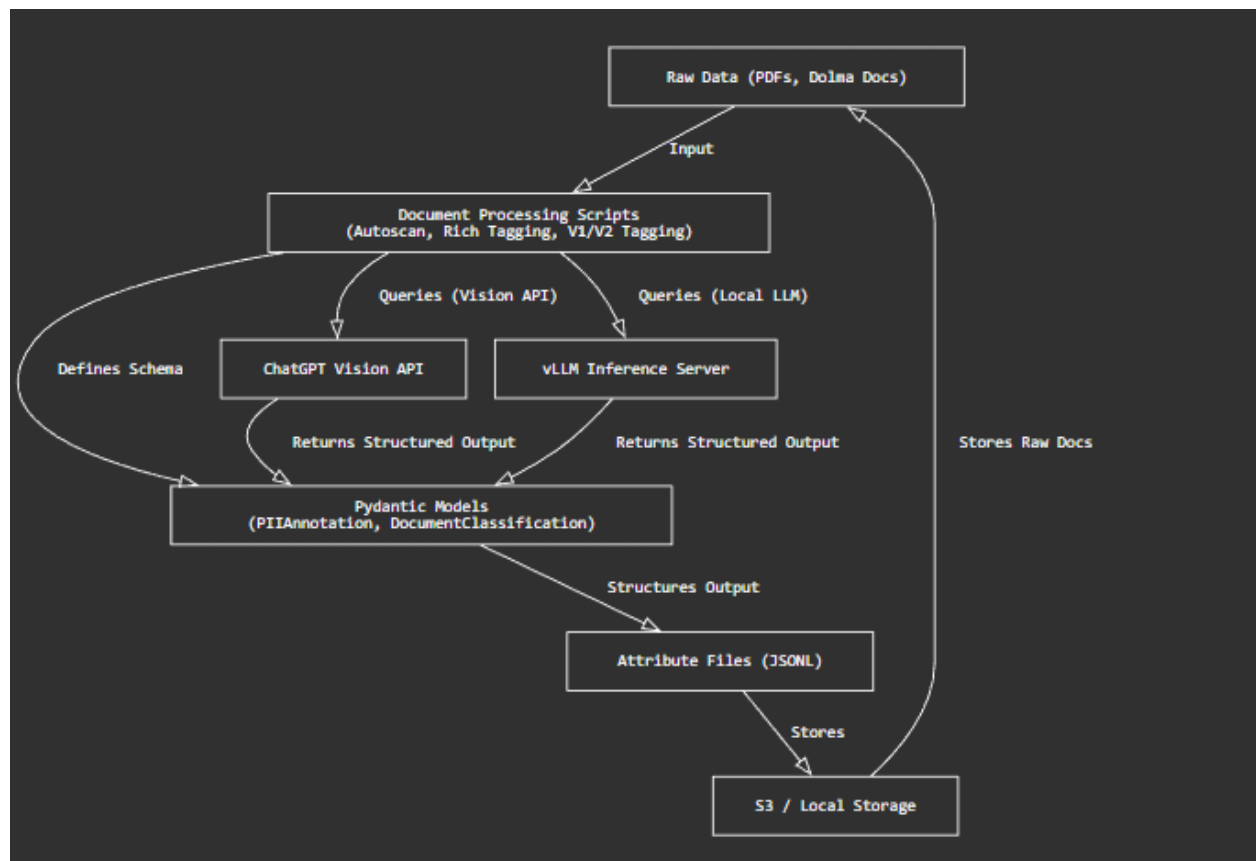
In addition to PII detection, the framework also supports document type classification. This involves analyzing PDF documents to determine their category (e.g., academic paper, news

article) and language, storing these classifications as attributes within the Dolma workspace. This classification can be performed by the same vision models used for PII detection. For instance, [scripts/pii/chatgpt\\_tag\\_dolmadocs\\_v2.py](#) focuses on classifying PDF documents to determine document types and language using the ChatGPT Vision API, storing these classifications as attributes.

The framework further includes mechanisms for comparing and evaluating different PII detection rules. This involves assessing the performance of a proposed hypothesis rule against a reference rule, calculating metrics such as Intersection over Union (IoU), precision, recall, and F1-score. The results of these comparisons are presented in comprehensive HTML reports and Cumulative Distribution Function (CDF) plots, which visualize rule performance and highlight discrepancies. The script [scripts/pii/pii\\_rule\\_comparison.py](#) is central to this rule comparison, supporting complex boolean rule expressions and generating interactive HTML reports. Orchestration scripts like [scripts/pii/check\\_qual.sh](#) facilitate the execution of these comparison tools.

For processing at scale, the system supports asynchronous and parallel execution, often utilizing platforms like Beaker for job submission and resource orchestration. This allows for efficient processing of large datasets by managing work queues and coordinating interactions with language model servers. Scripts like [scripts/pii/tagging\\_pipeline.py](#), [scripts/pii/rich\\_tagging\\_pipeline.py](#), and [scripts/pii/tagging\\_pipeline\\_v2.py](#) exemplify these capabilities, processing Dolma JSONL datasets for PII tagging, collecting responses from language models, and writing the results as attribute JSONL files while supporting both local and S3 storage. These pipelines often incorporate custom asynchronous HTTP clients and semaphore-based concurrency control to optimize interactions with model servers, as detailed in [Specialized GRPO Training and Multi-GPU Orchestration](#).

## AI Model Integration for PII and Document Classification



The olmOCR project integrates AI models to detect Personally Identifiable Information (PII) and classify document types, leveraging both OpenAI's ChatGPT Vision API and local vllm servers. This integration focuses on structured output using Pydantic models and handling diverse content types, including image-based PDF pages and raw text.

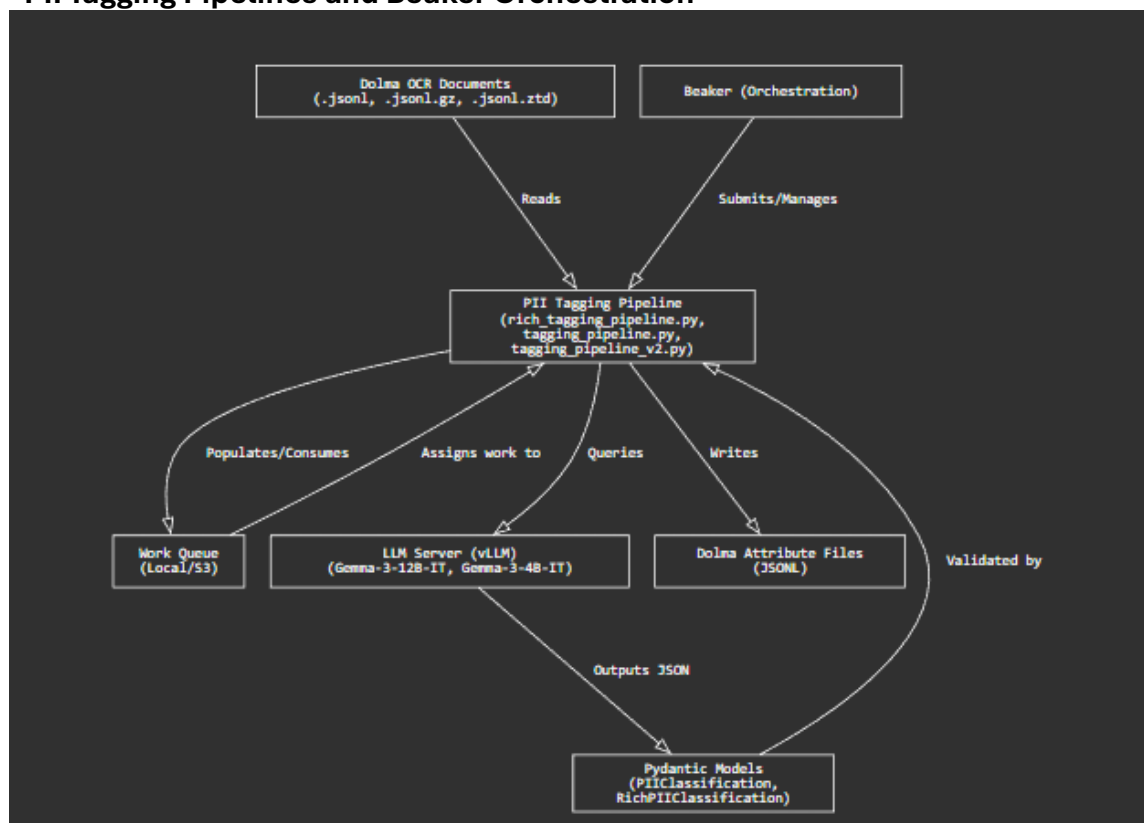
For PII detection and document classification, visual language models (VLMs) are utilized to analyze document content. Scripts such as `scripts/pii/autoscan_dolmadocs.py`, `scripts/pii/chatgpt_tag_dolmadocs_v1.py`, and `scripts/pii/chatgpt_tag_dolmadocs_v2.py` interact with the ChatGPT Vision API. These scripts convert PDF pages into base64 encoded PNG images to provide visual context to the AI model. For instance, `scripts/pii/autoscan_dolmadocs.py` processes random document pages, categorizes PII annotations (e.g., presence of names, email addresses), and generates a detailed report. Similarly, `scripts/pii/chatgpt_tag_dolmadocs_v2.py` focuses on classifying document types, such as academic papers or news articles, and identifying the language of a

document. These classifications are then stored as attributes within the Dolma document structure.

To ensure consistent and validated output from these AI models, Pydantic models like `PIIAnnotation` and `DocumentClassification` are employed. These models define the expected schema for the AI's responses, including boolean flags for various PII types or document categories, language codes, and document descriptions. This structured approach simplifies the integration of AI-generated insights into the overall data processing pipeline.

Beyond OpenAI's API, the framework also integrates with local vllm servers, as seen in pipelines like `scripts/pii/tagging_pipeline.py`, `scripts/pii/rich_tagging_pipeline.py`, and `scripts/pii/tagging_pipeline_v2.py`. These pipelines process Dolma JSONL datasets by sending text segments to a local language model for classification. The `PIIClassification` and `RichPIIClassification` Pydantic models are used here to define the schema for the language model's output, capturing a comprehensive range of PII categories. The local server integration allows for greater control over the inference process and can be orchestrated on platforms like Beaker for scalable execution. These pipelines emphasize asynchronous processing to handle high volumes of documents efficiently.

### PII Tagging Pipelines and Beaker Orchestration



The [olmocr](#) project includes a set of pipelines for identifying and classifying Personally Identifiable Information (PII) within Dolma OCR documents. These pipelines are designed to process large datasets efficiently, leveraging asynchronous programming, robust error handling, and scalable execution on platforms like Beaker.

The core of the PII tagging process involves utilizing language models, specifically OpenAI's ChatGPT Vision API or local [vllm](#) model servers, to analyze text and visual context from document pages. This analysis identifies various types of PII and classifies document types, such as academic papers or news articles. The results are structured into Pydantic models like [PIIClassification](#) or [RichPIIClassification](#), which define detailed schemas for the model's output, including language, document type, and specific PII categories like names, email addresses, or financial information.

Several Python scripts implement these tagging pipelines. For instance, [scripts/pii/tagging\\_pipeline.py](#), [scripts/pii/rich\\_tagging\\_pipeline.py](#), and [scripts/pii/tagging\\_pipeline\\_v2.py](#) are designed to process Dolma JSONL datasets. They operate by extracting text segments from documents, constructing detailed prompts for the language models, and sending these prompts to the model servers. These scripts employ asynchronous processing using [asyncio](#) to manage concurrent requests to the model, which is critical for handling large volumes of data. A custom asynchronous HTTP client, [apost](#), is used for interacting with the model servers, chosen for its control over low-level network interactions to prevent deadlocks at high request volumes. The pipelines also incorporate semaphores to regulate the number of parallel requests, ensuring that the model server is not overwhelmed while maintaining high GPU utilization.

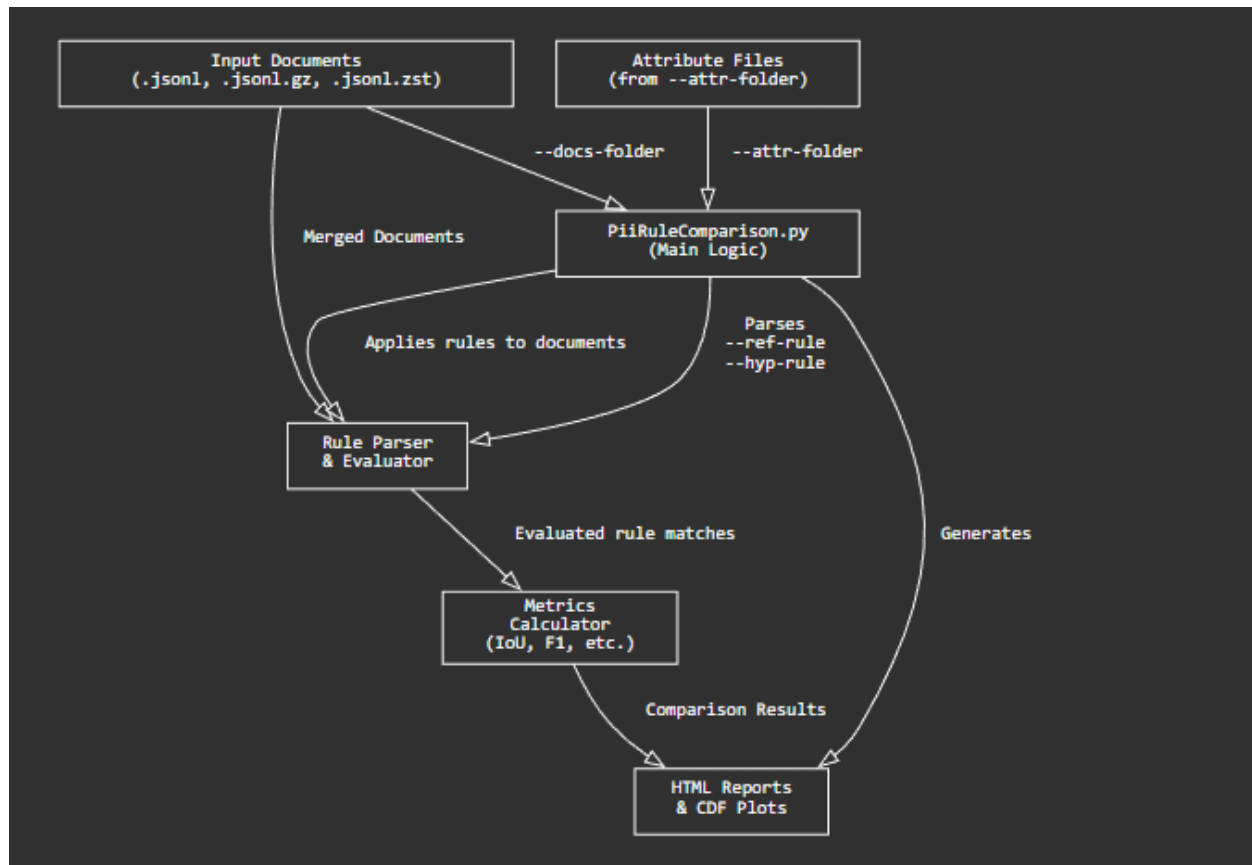
The results of this classification are then written back into the Dolma workspace as attribute JSONL files, mirroring the original document structure. This allows the identified PII and document type classifications to be associated directly with the source documents for downstream analysis or filtering.

Orchestration and scalable execution are achieved through integration with the Beaker platform. Scripts like [scripts/pii/run\\_tagging\\_pipeline.sh](#) facilitate submitting the entire PII tagging workflow as jobs to Beaker. This integration handles environment setup, secret management for credentials (e.g., AWS, Hugging Face), resource allocation (including GPU resources), and data synchronization from S3. This allows the pipelines to scale efficiently across distributed computing resources, enabling the processing of vast document collections.

Beyond tagging, the project also includes tools for comparing different PII detection rules. The script [scripts/pii/pii\\_rule\\_comparison.py](#) calculates metrics such as Intersection

over Union (IoU), precision, recall, and F1-score between a reference rule and a hypothesis rule. It generates interactive HTML reports and Cumulative Distribution Function (CDF) plots to visualize rule performance and identify differences, providing a framework for evaluating and refining PII detection strategies. For more information on comparing rules, see [PII Rule Comparison and Evaluation Framework](#).

## PII Rule Comparison and Evaluation Framework



A framework exists for comparing and evaluating different PII (Personally Identifiable Information) detection rules, or more generally, any document classification rules. This framework assesses the performance of a proposed "hypothesis" rule against a known "reference" rule by calculating standard metrics such as Intersection over Union (IoU), precision, recall, and F1-score.

The core of this evaluation is based on comparing the sets of documents identified by each rule. The system loads documents and their associated attributes from specified storage

locations, which can include both local file systems and S3 buckets, and can handle various compression formats. The attributes are then merged into the document objects, providing a unified structure for rule application.

Rules themselves are expressed as complex boolean expressions, allowing for flexible and detailed criteria. These expressions can combine multiple conditions using logical operators (AND, OR, NOT) and can involve checks on both boolean and numeric attributes. For instance, a rule might specify that a document must contain a certain type of PII and have an average numeric attribute value above a given threshold.

Once documents are evaluated against both the reference and hypothesis rules, the framework calculates the number of true positives, true negatives, false positives, and false negatives. These counts are then used to derive the IoU, precision, recall, and F1-score, providing a quantitative measure of how closely the hypothesis rule aligns with the reference rule.

To aid in analysis, the framework generates comprehensive HTML reports. These reports visualize the performance metrics and allow for interactive exploration of the documents. Users can navigate through documents classified as true positives, true negatives, false positives, and false negatives. Additionally, for numeric attributes, the system can generate Cumulative Distribution Function (CDF) plots, which are embedded within the HTML reports. These plots offer insights into the distribution of attribute values and help in understanding the impact of different thresholds or conditions within the rules. This overall process is orchestrated by scripts like `scripts/pii/pii_rule_comparison.py`, which can be invoked by shell scripts such as `scripts/pii/check_qual.sh`.