Pipelines

Pipelines in DocETL are the core structures that define the flow of data processing. They orchestrate the application of operators to datasets, creating a seamless workflow for complex chunk processing tasks.

Components of a Pipeline

A pipeline in DocETL consists of five main components:

1. **Default Model**: The language model to use for the pipeline.
2. **System Prompts**: A description of your dataset and the "persona" you'd like the LLM to adopt when analyzing your data.
3. **Datasets**: The input data sources for your pipeline.
4. **Operators**: The processing steps that transform your data.
5. **Pipeline Specification**: The sequence of steps and the output configuration.

Default Model

You can set the default model for a pipeline in the YAML configuration file. If no model is specified at the operation level, the default model will be used.

You can also tell DocETL to skip the dataset-level cache for the entire pipeline by enabling `bypass_cache`. When set to `true`, DocETL will neither read from nor write to its cache for any operation in that pipeline—which is helpful when you want to force fresh executions during development or debugging.

```yaml
default_model: gpt-4o-mini
bypass_cache: true  # optional - defaults to false
```

`bypass_cache` can still be overridden at the operator level if required.

You can also specify default API base URLs for language models and embeddings if you're hosting your own models with an OpenAI-compatible API:

> **Note**
>
> If you're hosting your own models with an OpenAI-compatible API, you can specify the base URLs:
>
> ```
> default_lm_api_base: https://your-custom-llm-endpoint.com/v1
> default_embedding_api_base: https://your-custom-embedding-endpoint.com/v1
> ```
>
> This is particularly useful when working with self-hosted models or services like Ollama, LM Studio, or other API-compatible LLM servers.

System Prompts

System prompts provide context to the language model about the data it's processing and the role it should adopt. This helps guide the model's responses to be more relevant and domain-appropriate. There are two key components to system prompts in DocETL:

1. **Dataset Description**: A concise explanation of what kind of data the model will be processing.
2. **Persona**: The role or perspective the model should adopt when analyzing the data.

Here's an example of how to define system prompts in your pipeline configuration:

```
system_prompt:
  dataset_description: a collection of transcripts of doctor visits
  persona: a medical practitioner analyzing patient symptoms and reactions to
medications
```

Datasets

Datasets define the input data for your pipeline. They are collections of items/chunks, where each item/chunk is an object in a JSON list (or row in a CSV file). Datasets are typically specified in the YAML configuration file, indicating the type and path of the data source. For example:

```
datasets:
  user_logs:
    type: file
    path: "user_logs.json"
```

Dynamic Data Loading

DocETL supports dynamic data loading, allowing you to process various file types by specifying a key that points to a path or using a custom parsing function. This feature is

particularly useful for handling diverse data sources, such as audio files, PDFs, or any other non-standard format.

To implement dynamic data loading, you can use parsing tools in your dataset configuration. Here's an example:

```yaml
datasets:
  audio_transcripts:
    type: file
    source: local
    path: "audio_files/audio_paths.json"
    parsing_tools:
      - input_key: audio_path
        function: whisper_speech_to_text
        output_key: transcript
```

In this example, the dataset configuration specifies a JSON file (audio_paths.json) that contains paths to audio files. The parsing_tools section defines how to process these files:

- `input_key`: Specifies which key in the JSON contains the path to the audio file. In this example, each object in the dataset should have a "audio_path" key, that represents a path to an audio file or mp3.
- `function`: Names the parsing function to use (in this case, the built-in whisper_speech_to_text function for audio transcription).
- `output_key`: Defines the key where the processed data (transcript) will be stored. You can access this in the pipeline in any prompts with the `{{ input.transcipt }}` syntax.

This approach allows DocETL to dynamically load and process various file types, extending its capabilities beyond standard JSON or CSV inputs. You can use built-in parsing tools or define custom ones to handle specific file formats or data processing needs. See the Custom Parsing documentation for more details.

> **Note**
>
> Currently, DocETL only supports JSON files or CSV files as input datasets. If you're interested in support for other data types or cloud-based datasets, please reach out to us or join our open-source community and contribute! We welcome new ideas and contributions to expand the capabilities of DocETL.

Dataset Description and Persona

You can define a description of your dataset and persona you want the LLM to adopt when executing operations on your dataset. This is useful for providing context to the

LLM and for optimizing the operations.

```
system_prompt: # This is optional, but recommended for better performance. It
is applied to all operations in the pipeline.
  dataset_description: a collection of transcripts of doctor visits
  persona: a medical practitioner analyzing patient symptoms and reactions to
medications
```

Operators

Operators are the building blocks of your pipeline, defining the transformations and analyses to be performed on your data. They are detailed in the Operators documentation. Operators can include map, reduce, filter, and other types of operations.

Pipeline Specification

The pipeline specification outlines the sequence of steps to be executed and the final output configuration. It typically includes:

- Steps: The sequence of operations to be applied to the data.
- Output: The configuration for the final output of the pipeline.

For example:

```
pipeline:
  steps:
    - name: analyze_user_logs
      input: user_logs
      operations:
        - extract_insights
        - unnest_insights
        - summarize_by_country
  output:
    type: file
    path: "country_summaries.json"
    intermediate_dir: "intermediate_data" # Optional: If you want to store
intermediate outputs in a directory
```

For a practical example of how these components come together, refer to the Tutorial, which demonstrates a complete pipeline for analyzing user behavior data.