Python API

The DocETL Python API provides a programmatic way to define, optimize, and run document processing pipelines. This approach offers an alternative to the YAML configuration method, allowing for more dynamic and flexible pipeline construction.

Overview

The Python API consists of several classes:

- Dataset: Represents a dataset with a type and path.
- Various operation classes (e.g., MapOp, ReduceOp, FilterOp) for different types of data processing steps.
- PipelineStep: Represents a step in the pipeline with input and operations.
- Pipeline: The main class for defining and running a complete document processing pipeline.
- PipelineOutput: Defines the output configuration for the pipeline.

Example Usage

Here's an example of how to use the Python API to create and run a simple document processing pipeline:

```
from docetl.api import Pipeline, Dataset, MapOp, ReduceOp, PipelineStep,
PipelineOutput

# Define datasets
datasets = {
    "my_dataset": Dataset(type="file", path="input.json", parsing=
[{"input_key": "file_path", "function": "txt_to_string", "output_key":
"content"}]),
}

# Note that the parsing is applied to the `file_path` key in each item of the
dataset,
# and the result is stored in the `content` key.

# Define operations
operations = [
    MapOp(
        name="process",
        type="map",
```

```
        prompt="Determine what type of document this is: {{ input.content }}",
        output={"schema": {"document_type": "string"}}
    ),
    ReduceOp(
        name="summarize",
        type="reduce",
        reduce_key="document_type",
        prompt="Summarize the processed contents: {% for item in inputs %}{{
item.content }} {% endfor %}",
        output={"schema": {"summary": "string"}}
    )
]

# Define pipeline steps
steps = [
    PipelineStep(name="process_step", input="my_dataset", operations=
["process"]),
    PipelineStep(name="summarize_step", input="process_step", operations=
["summarize"])
]

# Define pipeline output
output = PipelineOutput(type="file", path="output.json")

# Create the pipeline
pipeline = Pipeline(
    name="example_pipeline",
    datasets=datasets,
    operations=operations,
    steps=steps,
    output=output,
    default_model="gpt-4o-mini"
)

# Optimize the pipeline
optimized_pipeline = pipeline.optimize()

# Run the optimized pipeline
result = optimized_pipeline.run() # Saves the result to the output path

print(f"Pipeline execution completed. Total cost: ${result:.2f}")
```

This example demonstrates how to create a simple pipeline that processes input documents and then summarizes the processed content. The pipeline is optimized before execution to improve performance.

API Reference

For a complete reference of all available classes and their methods, please refer to the Python API Reference.

The API Reference provides detailed information about each class, including:

- Available parameters
- Method signatures
- Return types
- Usage examples