Python API Examples

This document provides two examples of how to use DocETL's Python API. Each example demonstrates a single pipeline step with multiple operations.

Example 1: Extract and Summarize Product Review Themes

This example extracts themes and quotes from product reviews, then summarizes the top themes across ALL documents using the special `_all` reduce key:

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

# Define dataset - A CSV of product reviews
dataset = Dataset(
    type="file",
    path="product_reviews.csv"  # Contains columns: review_id, product_id,
rating, review_text
)

# Define operations
operations = [
    # Extract themes and quotes from each review
    MapOp(
        name="extract_themes",
        type="map",
        prompt="""
        Analyze this product review and extract the key themes and
representative quotes:

        Review: {{ input.review_text }}
        Rating: {{ input.rating }}

        Identify 2-3 major themes (e.g., usability, quality, value) and
extract direct quotes that best represent each theme.
        """,
        output={
            "schema": {
                "themes": "list[string]",
                "quotes": "list[string]",
                "sentiment": "string"
            }
        }
    ),
    # Summarize all themes across reviews using _all key
    ReduceOp(
        name="summarize_themes",
        type="reduce",
        reduce_key="_all",  # Special key to reduce across all items
        prompt="""
        Analyze and synthesize the themes and quotes from these product
reviews:

        {% for item in inputs %}
        Review ID: {{ item.review_id }}
        Product: {{ item.product_id }}
        Rating: {{ item.rating }}
        Themes: {{ item.themes | join(", ") }}
        Quotes:
        {% for quote in item.quotes %}
        - "{{ quote }}"
        {% endfor %}
        Sentiment: {{ item.sentiment }}

        {% endfor %}

        Summarize the most frequent themes, the most representative quotes for
each theme, and the overall sentiment.
        """,
        output={
            "schema": {
                "summary": "string"
            }
        }
    )
]

# Define pipeline step (can consist of multiple operations)
step = PipelineStep(
    name="review_analysis",
    input="product_reviews",
    operations=["extract_themes", "summarize_themes"]
)

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

# Create and run pipeline
pipeline = Pipeline(
    name="review_analysis_pipeline",
    datasets={"product_reviews": dataset},
    operations=operations,
    steps=[step],
    output=output,
    default_model="gpt-4o-mini"
)

# Run the pipeline
cost = pipeline.run()
print(f"Pipeline execution completed. Total cost: ${cost:.2f}")
```

Example 2: Map-Unnest-Resolve-Reduce on Theme Keys

This example extracts theme-quote pairs from reviews, unnests them, resolves similar themes, and then reduces on the theme key. Here we will also optimize the pipeline.

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

# Define dataset - A JSON file with product reviews
dataset = Dataset(
    type="file",
    path="product_reviews.csv" # Same csv as previously
)

# Define operations
operations = [
    # Extract theme-quote pairs from each review
    MapOp(
        name="extract_theme_quotes",
        type="map",
        prompt="""
        Extract theme and quote pairs from this product review:

        Review: {{ input.review_text }}
        Product: {{ input.product_name }}
        Rating: {{ input.rating }}

        For each distinct theme in the review, extract a direct quote that
best represents that theme.
        Return each theme and its representative quote as a separate object in
the "theme_quotes" array.
        """,
        output={
            "schema": {
                "theme_quotes": "array"  # Array of objects with theme and
quote properties
            }
        }
    ),
    # Unnest to create separate items for each theme-quote pair
    UnnestOp(
        name="unnest_theme_quotes",
        type="unnest",
        array_path="theme_quotes"
    ),
    # Resolve similar themes using fuzzy matching
    ResolveOp(
        name="resolve_themes",
        type="resolve",
        comparison_prompt="""
        Determine if these two themes are the same or closely related:

        Theme 1: {{ input1.theme }}
        Theme 2: {{ input2.theme }}
        Consider semantic similarity, synonyms, and conceptual overlap.
        """,
        resolution_prompt="""
        Given the following list of similar themes, determine a canonical name
that best represents all of them:

        {% for item in inputs %}
        Theme: {{ item.theme }}
        {% endfor %}

        Choose a clear, concise name that accurately captures the core concept
shared across all these related themes.
        """
    ),
    # Reduce by theme to aggregate quotes and insights
    ReduceOp(
        name="aggregate_by_theme",
        type="reduce",
        reduce_key="theme",
        prompt="""
        Analyze all quotes related to the theme "{{ reduce_key }}":

        {% for item in inputs %}
        Product: {{ item.product_name }}
        Rating: {{ item.rating }}
        Quote: "{{ item.quote }}"

        {% endfor %}

        Summarize the key insights about this theme across all products and
ratings.
        """,
        output={
            "schema": {
                "summary": "string"
            }
        }
    )
]

# Define pipeline with a single step
step = PipelineStep(
    name="theme_analysis",
    input="product_reviews",
    operations=["extract_theme_quotes", "unnest_theme_quotes",
"resolve_themes", "aggregate_by_theme"]
)

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

# Create the pipeline
pipeline = Pipeline(
    name="theme_analysis_pipeline",
    datasets={"product_reviews": dataset},
    operations=operations,
    steps=[step],
    output=output,
    default_model="gpt-4o"
)

# Optimize the pipeline before running
optimized_pipeline = pipeline.optimize()

# Run the optimized pipeline
cost = optimized_pipeline.run()
print(f"Pipeline execution completed. Total cost: ${cost:.2f}")
```

Note that datasets can be json or CSV.