Optimization

Sometimes, finding the optimal pipeline for your task can be challenging. You might wonder:

Questions

- Will a single LLM call suffice for your task?
- Do you need to decompose your task or data further for better results?

To address these questions and improve your pipeline's performance, you can use DocETL to build an optimized version of your pipeline.

The DocETL Optimizer

The DocETL optimizer is designed to decompose operators (and sequences of operators) into their own subpipelines, potentially leading to higher accuracy.

Example

A map operation attempting to perform multiple tasks can be decomposed into separate map operations, ultimately improving overall accuracy. For example, consider a map operation on student survey responses that tries to:

1. Extract actionable suggestions for course improvement
2. Identify potential interdisciplinary connections

This could be optimized into two *separate* map operations:

- Suggestion Extraction: Focus solely on identifying concrete, actionable suggestions for improving the course.

```
prompt: |
  From this student survey response, extract any specific, actionable
suggestions
  for improving the course. If no suggestions are present, output 'No
suggestions found.':
  '{{ input.response }}'
```

- Interdisciplinary Connection Analysis: Analyze the response for mentions of concepts or ideas that could connect to other disciplines or courses.

```
prompt: |
  Identify any concepts or ideas in this student survey response that could
have
  interdisciplinary connections. For each connection, specify the related
discipline or course:
  '{{ input.response }}'
```

By breaking these tasks into separate operations, each LLM call can focus on a specific aspect of the analysis. This specialization might lead to more accurate results, depending on the LLM, data, and nature of the task!

How It Works

The DocETL optimizer operates using the following mechanism:

1. **Generation and Evaluation Agents**: These agents generate different plans for the pipeline according to predefined rewrite rules. Evaluation agents then compare plans and outputs to determine the best approach.
2. **Operator Rewriting**: The optimizer looks through operators in your pipeline where you've set optimize: true, and attempts to rewrite them using predefined rules.
3. **Output**: After optimization, DocETL outputs a new YAML file representing the optimized pipeline.

Using the Optimizer

You can invoke the optimizer using the following command:

```
docetl build your_pipeline.yaml
```

This command will save the optimized pipeline to `your_pipeline_opt.yaml`. Note that the optimizer will only rewrite operators where you've set `optimize: true`. Leaving this field unset will skip optimization for that operator.