Schemas

In DocETL, schemas play an important role in defining the structure of output from LLM operations. Every LLM call in DocETL is associated with an output schema, which specifies the expected format and types of the output data.

Overview

- Schemas define the structure and types of output data from LLM operations.
- They help ensure consistency and facilitate downstream processing.
- DocETL uses structured outputs or tool API to enforce these schemas.

> **Schema Simplicity**
>
> We've observed that **the more complex the output schema is, the worse the quality of the output tends to be**. Keep your schemas as simple as possible for better results.

Defining Schemas

Schemas are defined in the `output` section of an operator. They support various data types:

Type	Aliases	Description
`string`	`str`, `text`, `varchar`	For text data
`integer`	`int`	For whole numbers
`number`	`float`, `decimal`	For decimal numbers
`boolean`	`bool`	For true/false values
`enum`	-	For a set of possible values

Type	Aliases	Description
`list`	-	For arrays or sequences of items (must specify element type)
Objects	-	Using notation `{field: type}`

Filter Operation Schemas

Filter operation schemas must have a boolean type output field. This is used to determine whether each item should be included or excluded based on the filter criteria.

Examples

Simple Schema

```yaml
output:
  schema:
    summary: string
    sentiment: string
    include_item: boolean # For filter operations
```

Complex Schema

```yaml
output:
  schema:
    insights: "list[{insight: string, confidence: number}]"
    metadata: "{timestamp: string, source: string}"
```

Lists and Objects

Lists in schemas must specify their element type:

- `list[string]` : A list of strings
- `list[int]` : A list of integers
- `list[{name: string, age: integer}]` : A list of objects

Objects are defined using curly braces and must have typed fields:

- `{name: string, age: integer, is_active: boolean}`

Complex List Example

```
output:
  schema:
    users: "list[{name: string, age: integer, hobbies: list[string]}]"
```

Make sure that you put the type in quotation marks, if it references an object type (i.e., has curly braces)! Otherwise the yaml won't compile!

Enum Types

You can also specify enum types, which will be validated against a set of possible values. Suppose we have an operation to extract sentiments from a document, and we want to ensure that the sentiment is one of the three possible values. Our schema would look like this:

```
output:
  schema:
    sentiment: "enum[positive, negative, neutral]"
```

You can also specify a list of enum types (say, if we wanted to extract *multiple* sentiments from a document):

```
output:
  schema:
    possible_sentiments: "list[enum[positive, negative, neutral]]"
```

How We Enforce Schemas

DocETL uses structured outputs or tool API to enforce schema typing. This ensures that the LLM outputs adhere to the specified schema, making the results more consistent and easier to process in subsequent operations.

DocETL supports two output modes that determine how the LLM generates structured outputs:

Tools Mode (Default)

Uses the OpenAI tools/function calling API to enforce schema structure. This is the default mode and provides robust schema validation.

```
output:
  schema:
```

```
    summary: string
    sentiment: string
  mode: "tools"  # Optional - this is the default
```

Structured Output Mode

Uses LiteLLM's structured output feature with JSON schema validation. This mode can provide more reliable schema adherence for complex outputs.

```
output:
  schema:
    insights: "list[{insight: string, confidence: number}]"
  mode: "structured_output"
```

> **When to Use Structured Output Mode**
>
> Consider using `structured_output` mode when:
>
> - You have complex nested schemas with lists and objects
> - You need more consistent schema adherence
> - You're experiencing schema validation issues with tools mode
>
> The tools mode remains the default and works well for most use cases.

Mode Configuration

The output mode can be configured in the `output` section of any operation:

```
operations:
  - name: analyze_text
    type: map
    prompt: "Analyze the following text..."
    output:
      schema:
        topics: "list[{topic: string, relevance: number}]"
      mode: "structured_output"  # or "tools"
    model: gpt-4o-mini
```

If no mode is specified, DocETL defaults to `"tools"` mode for backward compatibility.

Best Practices

1. Keep output fields simple and use string types whenever possible.

2. Only use structured fields (like lists and objects) when necessary for downstream analysis or reduce operations.
3. If you need to reference structured fields in downstream operations, consider breaking complex structures into multiple simpler operations.

Schema Optimization

If you find your schema becoming too complex, consider breaking it down into multiple operations. This can improve both the quality of LLM outputs and the manageability of your pipeline.

Breaking Down Complex Schemas

Instead of:

```yaml
output:
  schema:
    summary: string
    key_points: "list[{point: string, sentiment: string}]"
```

Consider:

```yaml
output:
  schema:
    summary: string
    key_points: "string"
```

Where in the prompt you can say something like: `In your key points, please include the sentiment of each point.`

The only reason to use the complex schema is if you need to do an operation at the point level, like resolve them and reduce on them.

By following these guidelines and best practices, you can create effective schemas that enhance the performance and reliability of your DocETL operations.