Split Operation

The Split operation in DocETL is designed to divide long text content into smaller, manageable chunks. This is particularly useful when dealing with large documents that exceed the token limit of language models or when the LLM's performance degrades with increasing input size for complex tasks.

Motivation

Some common scenarios where the Split operation is valuable include:

- Processing long customer support transcripts to analyze specific sections
- Dividing extensive research papers or reports for detailed analysis
- Breaking down large legal documents to extract relevant clauses or sections
- Preparing long-form content for summarization or topic extraction

🚀 Operation Example: Splitting Customer Support Transcripts

Here's an example of using the Split operation to divide customer support transcripts into manageable chunks:

```yaml
- name: split_transcript
  type: split
  split_key: transcript
  method: token_count
  method_kwargs:
    num_tokens: 500
    model: gpt-4o-mini
```

This Split operation processes long customer support transcripts:

1. Splits the 'transcript' field into chunks of approximately 500 tokens each.
2. Uses the gpt-4o-mini model's tokenizer for accurate token counting.
3. Generates multiple output items for each input item, one for each chunk.

Note that chunks will not overlap in content.

Configuration

Required Parameters

- `type`: Must be set to "split".
- `split_key`: The key of the field containing the text to split.
- `method`: The method to use for splitting. Options are "delimiter" and "token_count".
- `method_kwargs`: A dictionary of keyword arguments for the splitting method.
- For "delimiter" method: `delimiter` (string) to use for splitting.
- For "token_count" method: `num_tokens` (integer) specifying the maximum number of tokens per chunk.

Optional Parameters in `method_kwargs

Parameter	Description	Default
`model`	The language model's tokenizer to use	Falls back to `default_model`
`num_splits_to_group`	Number of splits to group together into one chunk (only for "delimiter" method)	1
`sample`	Number of samples to use for the operation	None

Splitting Methods

Token Count Method

The token count method splits the text into chunks based on a specified number of tokens. This is useful when you need to ensure that each chunk fits within the token limit of your language model, or you know that smaller chunks lead to higher performance.

Delimiter Method

The delimiter method splits the text based on a specified delimiter string. This is particularly useful when you want to split your text at logical boundaries, such as paragraphs or sections.

Delimiter Method Example

If you set the `delimiter` to `"\n\n"` (double newline) and `num_splits_to_group` to 3, each chunk will contain 3 paragraphs.

```
- name: split_by_paragraphs
  type: split
  split_key: document
  method: delimiter
  method_kwargs:
    delimiter: "\n\n"
  num_splits_to_group: 3
```

Output

The Split operation generates multiple output items for each input item:

- All original key-value pairs from the input item.
- `{split_key}_chunk`: The content of the split chunk.
- `{op_name}_id`: A unique identifier for each original document.
- `{op_name}_chunk_num`: The sequential number of the chunk within its original document.

Use Cases

1. **Analyzing Customer Frustration**: Split long support transcripts, then use a map operation to identify frustration indicators in each chunk, followed by a reduce operation to summarize frustration points across the chunks (per transcript).
2. **Document Summarization**: Split large documents, apply a map operation for section-wise summarization, then use a reduce operation to compile an overall summary.
3. **Topic Extraction from Research Papers**: Divide research papers into sections, use a map operation to extract key topics from each section, then apply a reduce operation to synthesize main themes across the entire paper.

🚀 End-to-End Pipeline Example: Analyzing Customer Frustration

Let's walk through a complete example of using Split, Map, and Reduce operations to analyze customer frustration in support transcripts.

Step 1: Split Operation

```yaml
- name: split_transcript
  type: split
  split_key: transcript
  method: token_count
  method_kwargs:
    num_tokens: 500
    model: gpt-4o-mini
```

Step 2: Map Operation (Identify Frustration Indicators)

```yaml
- name: identify_frustration
  type: map
  input:
    - transcript_chunk
  prompt: |
    Analyze the following customer support transcript chunk for signs of
customer frustration:

    {{ input.transcript_chunk }}

    Identify any indicators of frustration, such as:
    1. Use of negative language
    2. Repetition of issues
    3. Expressions of dissatisfaction
    4. Requests for escalation

    Provide a list of frustration indicators found, if any.
  output:
    schema:
      frustration_indicators: list[string]
```

Step 3: Reduce Operation (Summarize Frustration Points)

```yaml
- name: summarize_frustration
  type: reduce
  reduce_key: split_transcript_id
  associative: false
  prompt: |
    Summarize the customer frustration points for this support transcript:

    {% for item in inputs %}
    Chunk {{ item.split_transcript_chunk_num }}:
    {% for indicator in item.frustration_indicators %}
    - {{ indicator }}
    {% endfor %}
    {% endfor %}

    Provide a concise summary of the main frustration points and their
frequency or intensity across the entire transcript.
  output:
```

```
    schema:
      frustration_summary: string
      primary_issues: list[string]
      frustration_level: string # e.g., "low", "medium", "high"
```

Non-Associative Reduce Operation

Note the `associative: false` parameter in the reduce operation. This is crucial when the order of the chunks matters for your analysis. It ensures that the reduce operation processes the chunks in the order they appear in the original transcript, which is often important for understanding the context and progression of customer frustration.

Explanation

1. The **Split** operation divides long transcripts into 500-token chunks.
2. The **Map** operation analyzes each chunk for frustration indicators.
3. The **Reduce** operation combines the frustration indicators from all chunks of a transcript, summarizing the overall frustration points, primary issues, and assessing the overall frustration level. The `associative: false` setting ensures that the chunks are processed in their original order.

This pipeline allows for detailed analysis of customer frustration in long support transcripts, which would be challenging to process in a single pass due to token limitations or degraded LLM performance on very long inputs.

Best Practices

1. **Choose the Right Splitting Method**: Use the token count method when working with models that have strict token limits. Use the delimiter method when you need to split at logical boundaries in your text.
2. **Balance Chunk Size**: When using the token count method, choose a chunk size that balances between context preservation and model performance. Smaller chunks may lose context, while larger chunks may degrade model performance. The DocETL optimizer can find the chunk size that works best for your task, if you choose to use the optimizer.
3. **Consider Overlap**: In some cases, you might want to implement overlap between chunks to maintain context. This isn't built into the Split operation, but you can achieve it by post-processing the split chunks.
4. **Use Appropriate Delimiters**: When using the delimiter method, choose a delimiter that logically divides your text. Common choices include double newlines for

paragraphs, or custom markers for document sections. When using the delimiter method, adjust the `num_splits_to_group` parameter to create chunks that contain an appropriate amount of context for your task.

5. **Mind the Order**: If the order of chunks matters for your analysis, always set `associative: false` in your subsequent reduce operations.
6. **Optimize for Performance**: For very large documents, consider using a combination of delimiter and token count methods. First split into large sections using delimiters, then apply token count splitting to ensure no chunk exceeds model limits.

By leveraging the Split operation effectively, you can process large documents efficiently and extract meaningful insights using subsequent map and reduce operations.