# Map Operation

The Map operation in DocETL applies a specified transformation to each item in your input data, allowing for complex processing and insight extraction from large, unstructured documents.

## 🚀 Example: Analyzing Long-Form News Articles

Let's see a practical example of using the Map operation to analyze long-form news articles, extracting key information and generating insights.

```yaml
- name: analyze_news_article
  type: map
  prompt: |
    Analyze the following news article:
    "{{ input.article }}"

    Provide the following information:
    1. Main topic (1-3 words)
    2. Summary (2-3 sentences)
    3. Key entities mentioned (list up to 5, with brief descriptions)
    4. Sentiment towards the main topic (positive, negative, or neutral)
    5. Potential biases or slants in reporting (if any)
    6. Relevant categories (e.g., politics, technology, environment; list up
to 3)
    7. Credibility score (1-10, where 10 is highly credible)

  output:
    schema:
      main_topic: string
      summary: string
      key_entities: list[object]
      sentiment: string
      biases: list[string]
      categories: list[string]
      credibility_score: integer

  model: gpt-4o-mini
  validate:
    - len(output["main_topic"].split()) <= 3
    - len(output["key_entities"]) <= 5
    - output["sentiment"] in ["positive", "negative", "neutral"]
    - len(output["categories"]) <= 3
    - 1 <= output["credibility_score"] <= 10
  num_retries_on_validate_failure: 2
```

This Map operation processes long-form news articles to extract valuable insights:

1. Identifies the main topic of the article.
2. Generates a concise summary.
3. Extracts key entities (people, organizations, locations) mentioned in the article.
4. Analyzes the overall sentiment towards the main topic.
5. Identifies potential biases or slants in the reporting.
6. Categorizes the article into relevant topics.
7. Assigns a credibility score based on the content and sources.

The operation includes validation to ensure the output meets our expectations and will retry up to 2 times if validation fails.

## Sample Input and Output

Input:


```
[
  {
    "article": "In a groundbreaking move, the European Union announced
yesterday a comprehensive plan to transition all member states to 100%
renewable energy by 2050. The ambitious proposal, dubbed 'Green Europe 2050',
aims to completely phase out fossil fuels and nuclear power across the
continent.

    European Commission President Ursula von der Leyen stated, 'This is not
just about fighting climate change; it's about securing Europe's energy
independence and economic future.' The plan includes massive investments in
solar, wind, and hydroelectric power, as well as significant funding for
research into new energy storage technologies.

    However, the proposal has faced criticism from several quarters. Some
Eastern European countries, particularly Poland and Hungary, argue that the
timeline is too aggressive and could damage their economies, which are still
heavily reliant on coal. Industry groups have also expressed concern about the
potential for job losses in the fossil fuel sector.

    Environmental groups have largely praised the initiative, with Greenpeace
calling it 'a beacon of hope in the fight against climate change.' However,
some activists argue that the 2050 target is not soon enough, given the urgency
of the climate crisis.

    The plan also includes provisions for a 'just transition,' with billions of
euros allocated to retraining workers and supporting regions that will be most
affected by the shift away from fossil fuels. Additionally, it proposes
stricter energy efficiency standards for buildings and appliances, and
significant investments in public transportation and electric vehicle
infrastructure.

    Experts are divided on the feasibility of the plan. Dr. Maria Schmidt, an
energy policy researcher at the University of Berlin, says, 'While ambitious,
this plan is achievable with the right political will and technological
advancements.' However, Dr. John Smith from the London School of Economics
warns, 'The costs and logistical challenges of such a rapid transition should
not be underestimated.'

    As the proposal moves forward for debate in the European Parliament, it's
clear that 'Green Europe 2050' will be a defining issue for the continent in
the coming years, with far-reaching implications for Europe's economy,
environment, and global leadership in climate action."
  }
]
```


Output:


```
[
  {
    "main_topic": "EU Renewable Energy",
    "summary": "The European Union has announced a plan called 'Green Europe
2050' to transition all member states to 100% renewable energy by 2050. The
ambitious proposal aims to phase out fossil fuels and nuclear power, invest in
```

```
renewable energy sources, and includes provisions for a 'just transition' to
support affected workers and regions.",
    "key_entities": [
      {
        "name": "European Union",
        "description": "Political and economic union of 27 member states"
      },
      {
        "name": "Ursula von der Leyen",
        "description": "European Commission President"
      },
      {
        "name": "Poland",
        "description": "Eastern European country critical of the plan"
      },
      {
        "name": "Hungary",
        "description": "Eastern European country critical of the plan"
      },
      {
        "name": "Greenpeace",
        "description": "Environmental organization supporting the initiative"
      }
    ],
    "sentiment": "positive",
    "biases": [
      "Slight bias towards environmental concerns over economic impacts",
      "More emphasis on supportive voices than critical ones"
    ],
    "categories": [
      "Environment",
      "Politics",
      "Economy"
    ],
    "credibility_score": 8
  }
]
```


This example demonstrates how the Map operation can transform long, unstructured news articles into structured, actionable insights. These insights can be used for various purposes such as trend analysis, policy impact assessment, and public opinion monitoring.

## Required Parameters

- `name`: A unique name for the operation.
- `type`: Must be set to "map".

## Optional Parameters

| Parameter | Description | Default |
|---|---|---|
| `prompt` | The prompt template to use for the transformation. Access input variables with `input.keyname`. | None |
| `batch_prompt` | Template for processing multiple documents in a single prompt. Access batch with `inputs` list. | None |
| `max_batch_size` | Maximum number of documents to process in a single batch | None |
| `output.schema` | Schema definition for the output from the LLM. | None |
| `output.n` | Number of outputs to generate for each input. (only available for OpenAI models; this is used to generate multiple outputs from a single input and automatically turn into a bigger list) | 1 |
| `model` | The language model to use | Falls back to `default_model` |
| `optimize` | Flag to enable operation optimization | `True` |
| `recursively_optimize` | Flag to enable recursive optimization of operators synthesized as part of rewrite rules | `false` |
| `sample` | Number of samples to use for the operation | Processes all data |
| `tools` | List of tool definitions for LLM use | None |
| `validate` | List of Python expressions to validate the output | None |
| `flush_partial_results` | Write results of individual batches of map operation to disk for faster inspection | False |

| Parameter | Description | Default |
|---|---|---|
| `num_retries_on_validate_failure` | Number of retry attempts on validation failure | 0 |
| `gleaning` | Configuration for advanced validation and LLM-based refinement | None |
| `drop_keys` | List of keys to drop from the input before processing | None |
| `timeout` | Timeout for each LLM call in seconds | 120 |
| `max_retries_per_timeout` | Maximum number of retries per timeout | 2 |
| `timeout` | Timeout for each LLM call in seconds | 120 |
| `litellm_completion_kwargs` | Additional parameters to pass to LiteLLM completion calls. | {} |
| `skip_on_error` | If true, skip the operation if the LLM returns an error. | False |
| `bypass_cache` | If true, bypass the cache for this operation. | False |
| `pdf_url_key` | If specified, the key in the input that contains the URL of the PDF to process. | None |
| `calibrate` | Improve consistency across documents by using sample data as reference anchors. | False |
| `num_calibration_docs` | Number of documents to use sample and generate outputs for, for calibration. | 10 |

Note: If `drop_keys` is specified, `prompt` and `output` become optional parameters.

**Validation and Gleaning**

For more details on validation techniques and implementation, see operators.

## Batch Processing

The Map operation supports processing multiple documents in a single prompt using the `batch_prompt` parameter. This can be more efficient than processing documents individually, especially for simpler tasks and shorter documents, especially when there are LLM call limits. However, larger batch sizes (even > 5) can lead to more incorrect results, so use this feature judiciously.

**Batch Processing Example**

```
- name: classify_documents
  type: map
  max_batch_size: 5  # Process up to 5 documents in a single LLM call
  batch_prompt: |
    Classify each of the following documents into categories (technology,
business, or science):

    {% for doc in inputs %}
    Document {{loop.index}}:
    {{doc.text}}
    {% endfor %}

    Provide a classification for each document.
  prompt: |
    Classify the following document:
    {{input.text}}
  output:
    schema:
      category: string
```

When using batch processing:

1. The `batch_prompt` template receives an `inputs` list containing the batch of documents
2. Use `max_batch_size` to control how many documents are processed in each batch
3. You must also provide a `prompt` parameter that will be used in case the batch prompt's response cannot be parsed into the output schema
4. Gleaning and validation are applied to each document in the batch individually, after the batch has been processed by the LLM

**Batch Size Considerations**

Choose your `max_batch_size` carefully:

- Larger batches may be more efficient but risk hitting token limits
- Start with smaller batches (3-5 documents) and adjust based on your needs
- Consider document length when setting batch size

## Calibration for Consistency

The Map operation supports calibration to improve consistency across documents, especially for classification tasks or operations that require relative positioning (like rating scales). When enabled, calibration samples a subset of your documents, processes them with the original prompt, and then uses those results to generate reference anchors that help maintain consistency across all documents.

This is particularly useful for: - **Classification tasks** where documents need to be evaluated relative to each other - **Rating/scoring operations** where you want consistent scales - **Subjective judgments** that benefit from concrete examples

### Document Priority Classification with Calibration

Imagine you're processing a large collection of customer support tickets and want to classify them by priority. Without calibration, the LLM might be inconsistent - a "medium" priority ticket early in processing might be classified as "high" later when the LLM sees more severe issues.

```yaml
- name: classify_ticket_priority
  type: map
  calibrate: true  # Enable calibration
  num_calibration_docs: 15  # Use 15 tickets for calibration
  prompt: |
    Classify the following customer support ticket by priority level:

    Subject: {{ input.subject }}
    Description: {{ input.description }}
    Customer Tier: {{ input.customer_tier }}

    Classify as: low, medium, high, or critical
  output:
    schema:
      priority: string
      reasoning: string
  model: gpt-4o-mini
```

**How calibration works:**

1. **Sample**: Randomly selects 15 tickets from your dataset (using seed=42 for reproducibility)
2. **Process**: Runs the original prompt on these 15 tickets
3. **Analyze**: An LLM analyzes the sample results and generates reference anchors
4. **Augment**: Appends these reference anchors to your original prompt
5. **Execute**: Processes all tickets with the augmented prompt

**Example calibration output:**

```
# Original prompt gets augmented with something like:
#
# For reference, consider 'Server completely down for 500+ users' → critical as
your baseline for critical issues.
# Documents similar to 'Login button not working for one user' → low priority.
# For reference, consider 'Payment processing delays affecting checkout' → high
as your standard for high priority issues.
```

**When to Use Calibration**

Calibration is most beneficial when:

- Your task requires relative judgments (rating scales, classifications)
- You're processing documents that vary widely in characteristics
- Consistency across the entire dataset is more important than individual accuracy
- You have enough data for meaningful sampling (at least 20+ documents)

**Calibration Considerations**

- Adds a small overhead cost (processes calibration samples + calibration analysis)
- Uses a fixed seed (42) for reproducible sampling
- The calibration LLM call uses temperature=0.0 for consistent results

# Advanced Features

## PDF Processing

The Map operation can directly process PDFs using Claude or Gemini models. To use this feature:

1. Your input dataset must contain a key representing the URL of the PDF to process
2. Specify this field name using the `pdf_url_key` parameter in your map operation
3. The URLs must be publicly accessible or accessible to your environment

**PDF Processing Example**

Here's an example of processing a dataset of papers, where each paper is represented by a URL.

```
datasets:
  papers:
    type: file
    path: "papers/urls.json"  # Contains documents with PDF URLs

default_model: gemini/gemini-2.0-flash  # or claude models
operations:
  - name: extract_paper_info
    type: map
    pdf_url_key: url  # Tells DocETL which field contains the PDF URL
    prompt: |
      Summarize the paper.
    output:
      schema:
        paper_info: string

pipeline:
  steps:
    - name: extract_paper_info
      input: papers
      operations:
        - extract_paper_info
```

Your input data (`papers/urls.json`) should contain documents with PDF URLs:

```
[
  {"url": "https://assets.anthropic.com/m/1cd9d098ac3e6467/original/Claude-3-Model-Card.pdf"},
  ...
]
```

DocETL will: 1. Download each PDF 2. Extract the text content 3. Pass the content to the LLM with your prompt 4. Return the processed results:

```
[
  {
    "url": "https://assets.anthropic.com/m/1cd9d098ac3e6467/original/Claude-3-Model-Card.pdf",
    "paper_info": "This paper introduces Claude 3.5 Haiku and the upgraded Claude 3.5 Sonnet..."
  },
  ...
]
```

## Tool Use

Tools can extend the capabilities of the Map operation. Each tool is a Python function that can be called by the LLM during execution, and follows the OpenAI Function Calling API.

**Tool Definition Example**

```
tools:
- required: true
    code: |
    def count_words(text):
        return {"word_count": len(text.split())}
    function:
    name: count_words
    description: Count the number of words in a text string.
    parameters:
        type: object
        properties:
        text:
            type: string
        required:
        - text
```

**Warning**

Tool use and gleaning cannot be used simultaneously.

## Input Truncation

If the input doesn't fit within the token limit, DocETL automatically truncates tokens from the middle of the input data, preserving the beginning and end which often contain more important context. A warning is displayed when truncation occurs.

## Batching

If you have a really large collection of documents and you don't want to run them through the Map operation at the same time, you can use the `batch_size` parameter to process data in smaller chunks. This can significantly reduce memory usage and improve performance.

To enable batching in your map operations, you need to specify the `max_batch_size` parameter in your configuration.

```
- name: extract_summaries
  type: map
  max_batch_size: 5
  clustering_method: random
```

```
  prompt: |
    Summarize this text: "{{ input.text }}"
  output:
    schema:
      summary: string
```

In the above config, there will be no more than 5 API calls to the LLM at a time (i.e., 5 documents processed at a time, one per API call).

### Dropping Keys

You can use a map operation to act as an LLM no-op, and just drop any key-value pairs you don't want to save to the output file. To do this, you can use the `drop_keys` parameter.

```
- name: drop_keys_example
  type: map
  drop_keys:
    - "keyname1"
    - "keyname2"
```

## Best Practices

1. **Clear Prompts**: Write clear, specific prompts that guide the LLM to produce the desired output.
2. **Robust Validation**: Use validation to ensure output quality and consistency.
3. **Appropriate Model Selection**: Choose the right model for your task, balancing performance and cost.
4. **Optimize for Scale**: For large datasets, consider using `sample` to test your operation before running on the full dataset.
5. **Use Tools Wisely**: Leverage tools for complex calculations or operations that the LLM might struggle with. You can write any Python code in the tools, so you can even use tools to call other APIs or search the internet.

### Synthetic Data Generation

The Map operation supports generating multiple outputs for each input using the `output.n` parameter. This is particularly useful for synthetic data generation, content variations, or when you need multiple alternatives for each input item.

When you set `output.n` to a value greater than 1, the operation will: 1. Process each input once 2. Generate multiple outputs based on the same input 3. Return all generated outputs as separate items in the result list

This multiplies your dataset size by the factor of `n`.

### Synthetic Email Generation Example

Imagine you have a dataset of prospects and want to generate multiple email templates for each person. Here's how to generate 10 different email templates per prospect:

```yaml
datasets:
  prospects:
    type: file
    path: "prospects.json"  # Contains names, companies, roles, etc.

operations:
  - name: generate_email_templates
    type: map
    bypass_cache: true
    optimize: true
    output:
      n: 10  # Generate 10 unique emails per prospect
      schema:
        subject: "str"
        body: "str"
        call_to_action: "str"
    prompt: |
      Create a personalized cold outreach email for the following prospect:

      Name: {{ input.name }}
      Company: {{ input.company }}
      Role: {{ input.role }}
      Industry: {{ input.industry }}

      The email should:
      - Have a compelling subject line
      - Be brief (3-5 sentences)
      - Mention a specific pain point for their industry
      - Include a clear call to action
      - Sound natural and conversational, not sales-y

      Your response should be formatted as:
      Subject: [Your subject line]

      [Your email body]

      Call to action: [Your specific call to action]

pipeline:
  steps:
    - name: email_generation
      input: prospects
      operations:
        - generate_email_templates

  output:
    type: file
    path: "email_templates.json"
```

With this configuration, if your prospects.json file has 50 prospects, the output will contain 500 email templates (50 prospects × 10 emails each).

**Important Considerations**

- The `output.n` parameter is only available for OpenAI models
- Higher values of `n` will increase the cost of your operation proportionally
- For optimal results, keep your prompt focused and clear about generating diverse outputs
- When using pandas, the `n` parameter can be passed directly to the `map` method