Sample operation

The Sample operation in DocETL samples items from the input. It is meant mostly as a debugging tool:

Insert it before the last operation, the one you're currently trying to add to the end of a working pipeline, to limit the amount of data it will be fed, so that the run time is small enough to comfortably debug its prompt. Once it seems to be working, you can remove the sample operation. You can then repeat this for each operation you add while developing your pipeline!

🚀 Example:

```yaml
- name: sample_concepts
  type: sample
  method: uniform
  samples: 0.1
  stratify_key: category
  random_state: 42
```

This sample operation will return a pseudo-randomly selected 10% of the samples (samples: 0.1). The random selection will be seeded with a constant (42), meaning the same sample will be returned if you rerun the pipeline (If no random state is given, a different sample will be returned every time). Additionally, the random sampling will sample each value of the category key proportionally.

Required Parameters

- name: A unique name for the operation.
- type: Must be set to "sample".
- method: The sampling method to use. Can be "uniform", "outliers", "custom", "first", "top_embedding", or "top_fts".
- samples: Either a list of key-value pairs representing document ids and values, an integer count of samples, or a float fraction of samples.

Optional Parameters

Parameter	Description	Default
random_state	An integer to seed the random generator with	None
stratify_key	Key(s) to stratify by. Can be a string or list of strings	None
samples_per_group	When stratifying, sample N items per group vs. proportionally	False
method_kwargs	Additional parameters for specific methods (e.g., outliers)	{}

Sampling Methods

Uniform Sampling

Randomly samples items from the input data. When combined with stratification, maintains the distribution of the stratified groups.

```yaml
- name: uniform_sample
  type: sample
  method: uniform
  samples: 100
```

First Sampling

Takes the first N items from the input. When combined with stratification, takes proportionally from each group.

```yaml
- name: first_sample
  type: sample
  method: first
  samples: 50
```

Outlier Sampling

Samples based on distance from a center point in embedding space. Specify the following in method_kwargs:

- embedding_keys: A list of keys to use for creating embeddings.
- std: The number of standard deviations to use as the cutoff for outliers.

- samples: The number or fraction of samples to consider as outliers.
- keep: Whether to keep (true) or remove (false) the outliers. Defaults to false.
- center: (Optional) A dictionary specifying the center point for distance calculations.

You must specify either "std" or "samples" in the method_kwargs, but not both.

```yaml
- name: remove_outliers
  type: sample
  method: outliers
  method_kwargs:
    embedding_keys:
      - concept
      - description
    std: 2
    keep: false
```

Custom Sampling

Samples specific items by matching key-value pairs. Stratification is not supported with custom sampling.

```yaml
- name: custom_sample
  type: sample
  method: custom
  samples:
    - id: 1
    - id: 5
```

Top Embedding Sampling

Retrieves the top N most similar items to a query based on semantic similarity using embeddings. Requires the following in method_kwargs:

- keys: A list of keys to use for creating embeddings
- query: The query string to match against (supports Jinja templates)
- embedding_model: (Optional) The embedding model to use. Defaults to "text-embedding-3-small"

```yaml
- name: semantic_search
  type: sample
  method: top_embedding
  samples: 10
  method_kwargs:
    keys:
      - title
      - content
```

```
    query: "machine learning applications in healthcare"
    embedding_model: text-embedding-3-small
```

With Jinja template for dynamic queries:

```
- name: personalized_search
  type: sample
  method: top_embedding
  samples: 5
  method_kwargs:
    keys:
      - description
    query: "{{ input.user_query }}"
```

Top FTS Sampling

Retrieves the top N items using full-text search with BM25 algorithm. Requires the following in method_kwargs:

- keys: A list of keys to search within
- query: The query string for keyword matching (supports Jinja templates)

```
- name: keyword_search
  type: sample
  method: top_fts
  samples: 20
  method_kwargs:
    keys:
      - title
      - content
      - tags
    query: "python programming tutorial"
```

With dynamic query:

```
- name: search_products
  type: sample
  method: top_fts
  samples: 0.1  # Top 10% of results
  method_kwargs:
    keys:
      - product_name
      - description
    query: "{{ input.search_terms }}"
```

Stratification

Stratification can be applied to "uniform", "first", "outliers", "top_embedding", and "top_fts" methods. It ensures that the sample maintains the distribution of specified key(s) in the data or retrieves top items from each stratum.

Single Key Stratification

```yaml
- name: stratified_sample
  type: sample
  method: uniform
  samples: 0.2
  stratify_key: category
```

Multiple Key Stratification

When using multiple keys, stratification is based on the combination of values:

```yaml
- name: multi_stratified_sample
  type: sample
  method: uniform
  samples: 50
  stratify_key:
    - type
    - size
```

Samples Per Group

Instead of proportional sampling, you can sample a fixed number from each stratum:

```yaml
- name: stratified_per_group
  type: sample
  method: uniform
  samples: 10  # Sample 10 items from each group
  stratify_key: category
  samples_per_group: true
```

This also works with fractions:

```yaml
- name: stratified_fraction_per_group
  type: sample
  method: uniform
  samples: 0.3  # Sample 30% from each group
  stratify_key: category
  samples_per_group: true
```

Complete Examples

Stratified outlier detection:

```yaml
- name: stratified_outliers
  type: sample
  method: outliers
  stratify_key: document_type
  method_kwargs:
    embedding_keys:
      - title
      - content
    std: 1.5
    keep: false
```

Stratified first sampling with multiple keys:

```yaml
- name: stratified_first
  type: sample
  method: first
  samples: 100
  stratify_key:
    - category
    - priority
  samples_per_group: false  # Take proportionally from each combination
```

Outlier sampling with a custom center:

```yaml
- name: centered_outliers
  type: sample
  method: outliers
  method_kwargs:
    embedding_keys:
      - concept
      - description
    center:
      concept: Tree house
      description: A small house built among the branches of a tree for
children to play in.
    samples: 20  # Keep the 20 furthest items from the center
    keep: true
```

Stratified semantic search - retrieve top documents from each category:

```yaml
- name: stratified_semantic_search
  type: sample
  method: top_embedding
  samples: 5  # Get top 5 from each category
  stratify_key: category
  samples_per_group: true
  method_kwargs:
    keys:
      - title
      - abstract
    query: "recent advances in artificial intelligence"
```

Full-text search with multiple stratification keys:

```yaml
- name: stratified_keyword_search
  type: sample
  method: top_fts
  samples: 3
  stratify_key:
    - department
    - priority
  samples_per_group: true
  method_kwargs:
    keys:
      - subject
      - content
    query: "urgent customer complaint refund"
```

Note on TopK Operation

For retrieval use cases, consider using the dedicated TopK operation which provides a cleaner interface specifically designed for top-k retrieval with three methods: - `embedding`: Semantic similarity search - `fts`: Full-text search using BM25 - `llm_compare`: LLM-based ranking

The TopK operation offers the same functionality as the sample operation's `top_embedding` and `top_fts` methods, but with a more intuitive API for retrieval tasks.