Rank Operation

The Rank operation in DocETL sorts documents based on specified criteria. Note that this operation is designed to sort documents along some (latent) attribute in the data. **It is not specifically meant for top-k or retrieval-like queries.**

We adapt algorithms from Human-Powered Sorts and Joins (VLDB 2012).

🚀 Example: Ranking Debates by Level of Controversy

Let's see a practical example of using the Rank operation to rank political debates based on how controversial they are:

```yaml
- name: rank_by_controversy
  type: rank
  prompt: |
    Order these debate transcripts based on how controversial the discussion
is.
    Consider factors like:
    - The level of disagreement between candidates
    - Discussion of divisive topics
    - Strong emotional language
    - Presence of conflicting viewpoints
    - Public reaction mentioned in the transcript

    Debates with the most controversial content should be ranked highest.
  input_keys: ["content", "title", "date"]
  direction: desc
  rerank_call_budget: 10 # max number of LLM calls to use; also optional
  initial_ordering_method: "likert"
```

This Rank operation ranks debate transcripts from most controversial to least controversial by:

1. First generating ordinal scores (on the Likert scale) for the ranking criteria and each document. This executes an LLM call **per document**.
2. Creating an initial ranking based on the scores from step 1.
3. Using an LLM to perform more precise re-rankings on a sliding window of documents. This executes `rerank_call_budget` calls.

Sample Input and Output

Input:


```
[
  {
    "title": "Presidential Debate: Economy and Trade",
    "date": "2020-09-29",
    "content": "Moderator: Let's discuss trade policies. Candidate A, your
response?\n\nCandidate A: My opponent's policies have shipped jobs overseas for
decades! Our workers are suffering while other countries laugh at
us.\n\nCandidate B: That's simply not true. The data shows our export growth
has been strong. My opponent doesn't understand basic economics.\n\nCandidate
A: [interrupting] You've been in government for 47 years and haven't fixed
anything!\n\nCandidate B: If you'd let me finish... The manufacturing sector
has actually added jobs under our policies.\n\nModerator: Please allow each
other to finish. Let's move to healthcare..."
  },
  {
    "title": "Vice Presidential Debate: Foreign Policy",
    "date": "2020-10-07",
    "content": "Moderator: What would your administration's approach be to
China?\n\nCandidate C: We need strategic engagement that protects American
interests while avoiding unnecessary conflict. My opponent has proposed
policies that would damage our diplomatic relationships.\n\nCandidate D: I
respectfully disagree with my colleague. Our current approach has been too
soft. We need to stand firm on human rights issues and trade
imbalances.\n\nCandidate C: I think we actually agree on the goals, if not the
methods. The question is how to achieve them without harmful
escalation.\n\nCandidate D: That's a fair point. Perhaps there's a middle
ground that maintains pressure while keeping dialogue open.\n\nModerator: Thank
you both for that thoughtful exchange. Moving to the Middle East..."
  }
]
```


Output:


```
[
  {
    "title": "Presidential Debate: Economy and Trade",
    "date": "2020-09-29",
    "content": "Moderator: Let's discuss trade policies. Candidate A, your
response?\n\nCandidate A: My opponent's policies have shipped jobs overseas for
decades! Our workers are suffering while other countries laugh at
us.\n\nCandidate B: That's simply not true. The data shows our export growth
has been strong. My opponent doesn't understand basic economics.\n\nCandidate
A: [interrupting] You've been in government for 47 years and haven't fixed
anything!\n\nCandidate B: If you'd let me finish... The manufacturing sector
has actually added jobs under our policies.\n\nModerator: Please allow each
other to finish. Let's move to healthcare...",
    "_rank": 1
  },
  {
    "title": "Vice Presidential Debate: Foreign Policy",
    "date": "2020-10-07",
    "content": "Moderator: What would your administration's approach be to
China?\n\nCandidate C: We need strategic engagement that protects American
interests while avoiding unnecessary conflict. My opponent has proposed
```

```
policies that would damage our diplomatic relationships.\n\nCandidate D: I
respectfully disagree with my colleague. Our current approach has been too
soft. We need to stand firm on human rights issues and trade
imbalances.\n\nCandidate C: I think we actually agree on the goals, if not the
methods. The question is how to achieve them without harmful
escalation.\n\nCandidate D: That's a fair point. Perhaps there's a middle
ground that maintains pressure while keeping dialogue open.\n\nModerator: Thank
you both for that thoughtful exchange. Moving to the Middle East...",
    "_rank": 2
  }
]
```

This example demonstrates how the Rank operation can semantically sort documents based on complex criteria, providing a ranking that would be difficult to achieve with keyword matching or rule-based approaches.

Algorithm and Implementation

The Rank operation works in these steps:

1. **Initial Ranking**:
    a. The algorithm begins with either an embedding-based or Likert-scale rating approach:
        i. **Embedding-based**: Creates embedding vectors for the ranking criteria and each document, then calculates cosine similarity
        ii. **Likert-based** (default): Uses the LLM to rate each document on a 7-point Likert scale based on the criteria. We do this in batches of `batch_size` documents (defaults to 10), and the prompt includes a random sample of `num_calibration_docs` (defaults to 10) documents to calibrate the LLM with.
    b. Documents are initially sorted by their similarity scores or ratings (high to low for desc, low to high for asc)
2. **"Picky Window" Refinement**:
    a. Rather than processing all documents with equal focus, the algorithm employs a "picky window" approach
    b. Starting from the bottom of the currently ranked documents and working upward:
        i. A large window of documents is presented to the LLM
        ii. The LLM is asked to identify only the top few documents (configured via `num_top_items_per_window`)
        iii. These chosen documents are then moved to the beginning of the window
    c. The window slides upward through the document set with overlapping segments

   d. This approach enables the algorithm to process many documents while focusing LLM effort on identifying the best matches

3. **Efficient Resource Utilization**:
   a. The window size and step size are calculated based on the call budget to ensure optimal use of LLM calls
   b. Overlap between windows ensures robust ranking with minimal redundancy
   c. The algorithm tracks document positions using unique identifiers to maintain consistency
4. **Output Preparation**:
   a. After all windows have been processed, the algorithm assigns a `_rank` field to each document (1-indexed)
   b. Returns the documents in their final sorted order

Required Parameters

- `name`: A unique name for the operation.
- `type`: Must be set to "rank".
- `prompt`: The prompt specifying the ranking criteria. This does **not** need to be a Jinja template.
- `input_keys`: List of document keys to consider for ranking.
- `direction`: Either "asc" (ascending) or "desc" (descending).

Optional Parameters

Parameter	Description	Default
`model`	The language model to use for LLM-based ranking	Falls back to `default_model`
`embedding_model`	The embedding model to use for similarity calculations	"text-embedding-3-small"
`batch_size`	Maximum number of documents to process in a single LLM batch rating (used for the first pass)	10

Parameter	Description	Default
`timeout`	Timeout for each LLM call in seconds	120
`verbose`	Whether to log detailed LLM call statistics	False
`num_calibration_docs`	Number of documents to use for calibration (used for the first pass)	10
`litellm_completion_kwargs`	Additional parameters to pass to LiteLLM completion calls	{}
`bypass_cache`	If true, bypass the cache for this operation	False
`initial_ordering_method`	Method to use for initial ranking: "likert" (default) or "embedding"	"likert"
`k`	Number of top items to focus on in the final ranking	None (ranks all items)
`call_budget`	Maximum number of LLM API calls to make during ranking	10
`num_top_items_per_window`	Number of top items the LLM should select from each window	3
`overlap_fraction`	Fraction of overlap between windows	0.5

Two-Step Ranking Approach

For more complex ranking tasks, a two-step approach can be more effective:

1. First use a `map` operation to extract and structure relevant information
2. Then use the `rank` operation to rank based on the extracted information

Two-Step Ranking Example

```
operations:
  - name: extract_hostile_exchanges
    type: map
    output:
      schema:
        meanness_summary: "str"
        hostility_level: "int"
        key_examples: "list[str]"
    prompt: |
      Analyze the following debate transcript for {{ input.title }} on {{
input.date }}:

      {{ input.content }}

      Extract and summarize exchanges where candidates are mean or hostile to
each other.
      [... prompt details ...]

  - name: rank_by_meanness
    type: rank
    prompt: |
      Order these debate transcripts based on how mean or hostile the
candidates are to each other.
      Focus on the meanness summaries and examples that have been extracted.

      Consider:
      - The overall hostility level rating
      - Severity of personal attacks in the key examples
      [... prompt details ...]
    input_keys: ["meanness_summary", "hostility_level", "key_examples",
"title", "date"]
    direction: desc
    rerank_call_budget: 10

pipeline:
  steps:
    - name: meanness_analysis
      input: debates
      operations:
        - extract_hostile_exchanges
        - rank_by_meanness
```

This approach: 1. First extracts structured data about hostility in each debate 2. Then ranks debates based on this pre-processed data

Best Practices

1. **Craft Clear Ranking Criteria**: Write clear, specific prompts that guide the LLM to understand the ranking priorities.

2. **Choose Appropriate Input Keys**: Only include document fields that are relevant to the ranking criteria to reduce noise.
3. **Consider Pre-Processing**: For complex criteria, use a map operation first to extract structured data that makes ranking more effective.
4. **Tune Window Parameters**:
5. Adjust `num_top_items_per_window` based on how selective you need the ranking to be
6. Modify `overlap_fraction` to balance redundancy and completeness
7. Start with defaults and adjust based on results
8. **Use Verbose Mode During Development**: Enable the `verbose` flag during development to understand how the ranking process works and verify the results.
9. **Direction Matters**: Choose "asc" or "desc" carefully based on your use case:
10. "desc" (descending) ranks the most matching items first
11. "asc" (ascending) ranks the least matching items first
12. **Mind Cost Considerations**: The ranking operation makes multiple LLM calls and embedding requests. For large datasets, consider sampling first to test your approach. Use the embedding based first pass to significantly reduce cost.

Performance Considerations

- The rank operation scales with O(n)
- The `verbose` flag adds detailed logging but doesn't affect performance or results