Beyond Quacking: Deep Integration of Language Models and RAG into DuckDB

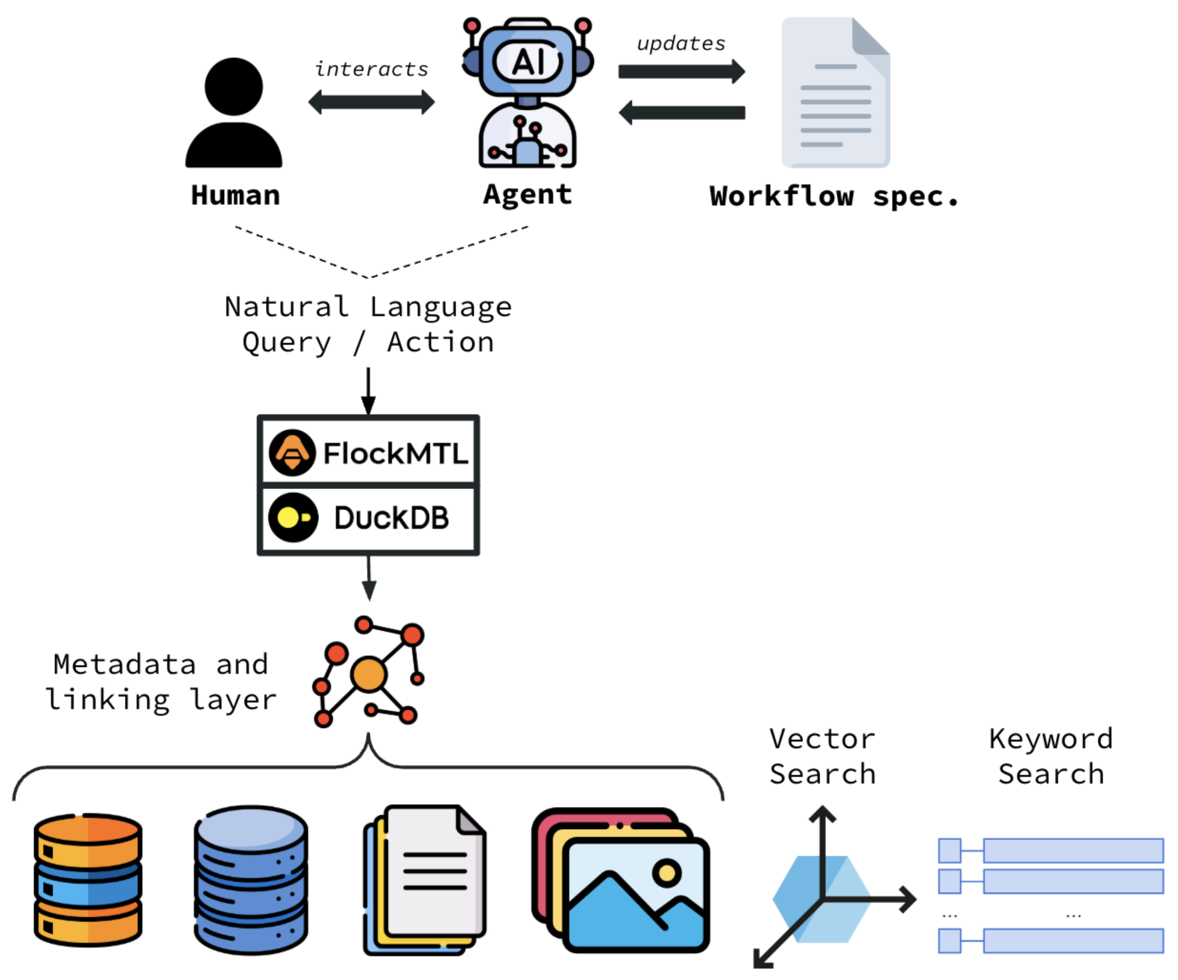
Anas Dorbani Sunny Yasser Jimmy Lin Amine Mhedhbi

Polytechnique Montréal

Overview

We designed **FlockMTL** for query processing on multi-modal data. Existing approaches face several challenges:

1. Implementations are ad hoc with many low-level decisions.
2. Reliance on loosely coupled systems and hand-written orchestrators.



Resource Independence

- Example creating a `MODEL` and a `PROMPT`:

```sql
CREATE GLOBAL MODEL('model-relevance-check', 'gpt-4o-mini', 'openai');
CREATE PROMPT('joins-prompt', 'is related to join algos given abstract');
```

- `MODEL`s and `PROMPT`s as **first-class SQL schema objects**.
- Run **CRUD** operations on the resources versioned.

Scalar Semantic Operations

- Example using scalar functions with text and images:

```sql
SELECT P.id, P.title, P.abstract, P.content
  FROM research_papers P
 WHERE llm_filter(
    {'model_name': 'model-relevance-check'},
    {'prompt_name': 'joins-prompt',
     'context_columns': [
        {'data': P.title, 'name': 'title'},
        {'data': P.abstract, 'name': 'abstract'}
    ]});
```

(a) Filtering by relevance to join algos.

```sql
SELECT llm_complete(
    {'model_name': 'gpt-4o'},
    {'prompt': 'Briefly describe audience, appeal, and one improvement.'
     'context_columns': [
        {'data': P.image_url, 'name': 'product_image', 'type': 'image'},
        {'data': P.product_name, 'name': 'product_name'},
        {'data': P.category, 'name': 'category'}
    ]}) AS product_analysis
  FROM product_images P;
```

(b) Identify product items from images.

Function	Description
`llm_complete`	generates text or structured output from an input tuple or an image.
`llm_filter`	returns `True`/`False` for a given tuple or an image.
`llm_embedding`	generates an embedding vector from an input text value.
`fusion`	fuses N normalized scores, e.g., using `rrf` / `combanz` / `combmed` / etc.

Unlocking Hybrid Search → RAG

- Example using data fusion to unlock hybrid search:

```sql
WITH BM25 AS (SELECT idx, text, norm_score FROM ...),
VS AS (SELECT idx, text, norm_score FROM ...),

-- Reciprocal rank fusion of BM25 and VS
passages_fused_scores AS (
  SELECT COALESCE(BM25.idx, VS.idx) AS idx, t.content,
         fusion_rrf(BM25.norm_score, VS.norm_score) AS fused_score
   FROM BM25 FULL OUTER JOIN VS USING (idx)
   JOIN my_table t USING (idx)
  ORDER BY fused_score DESC
  LIMIT 10
)

-- Use llm_reduce to generate a final answer from the top ranked passages
SELECT llm_reduce(content, 'Give me a full summary of the data')
FROM passages_fused_scores;
```

Aggregate Semantic Operations

- Example using aggregate functions summarizing research themes:

```sql
SELECT P. publication_year,
    llm_reduce(
    {'model': 'gpt-4o'},
    {'prompt': 'Summarize the key research themes',
      'context_columns': [
        {'data': P.abstract, 'name': 'abstract'}
    ]}
) AS year_summary
FROM research_papers P
GROUP BY P. publication_year;
```

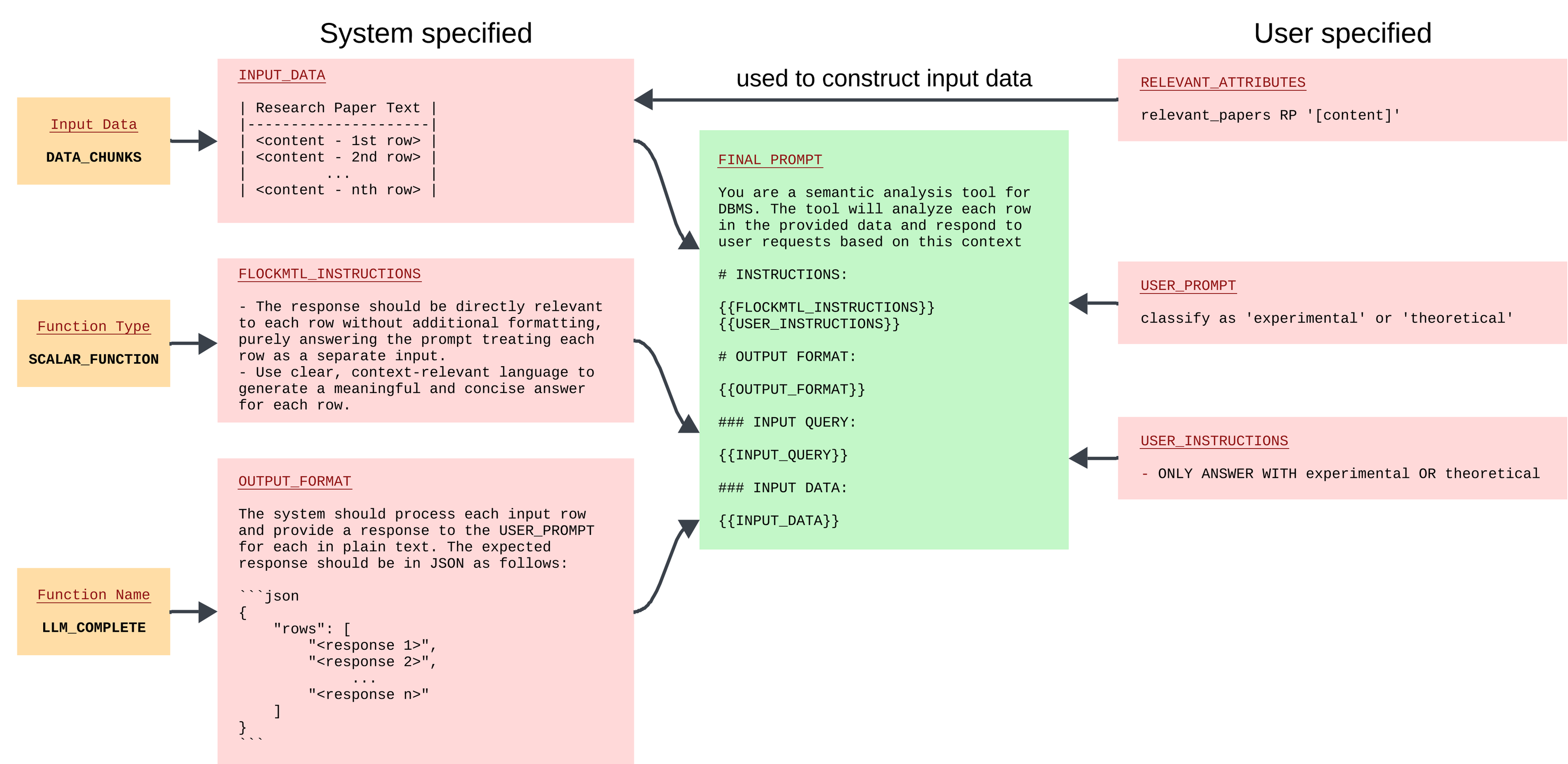
Function	Description
`llm_reduce`	generates text or structured output from multiple input tuples or images.
`llm_rerank`	ranks input tuples based on relevance.
`llm_first/last`	returns the most or least relevant tuple from multiple input tuples.

→ With **CTEs**, these functions allow interleaving **analytics** with **LLM predictions**, supporting semantic operations, e.g., **filtering**, **hybrid search**, **extraction**, and **ranking**. A use case of interest is *market intelligence*.

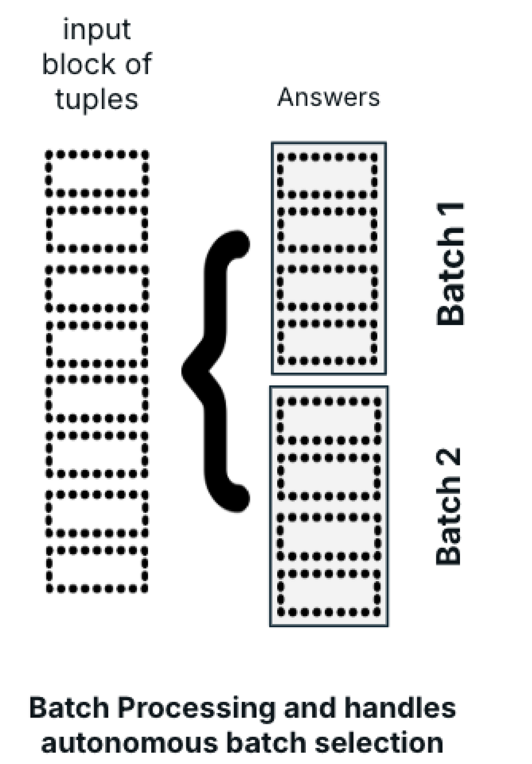
Built-in Optimizations

Showcase three optimizations: (i) **Meta-prompting**; (ii) **Batching**; and (iii) **Async execution**. Also explored **deduplication** and **caching**.

- **Meta-prompting:** Base prompt templated with the user prompt to reduce variability and improve LLM accuracy.



- **Batching:** Groups dynamically multiple input rows for an LLM prediction. Done per input data vector to a `Projection` operator to reduce latency, token usage and model API calls.



Batch Processing and handles autonomous batch selection

- **Async:** Send async prediction requests for batches of the data vector to a `Projection` operator instead of synchronously.

For the setup with 4 threads and batch size 32:

- **Sync + batching**: 196.04s for 1k tuples.
- **Async + batching**: 16s for 6k tuples; 6s for 3k tuples.
- **Sync without batching**: over 100× slower, often timing out.