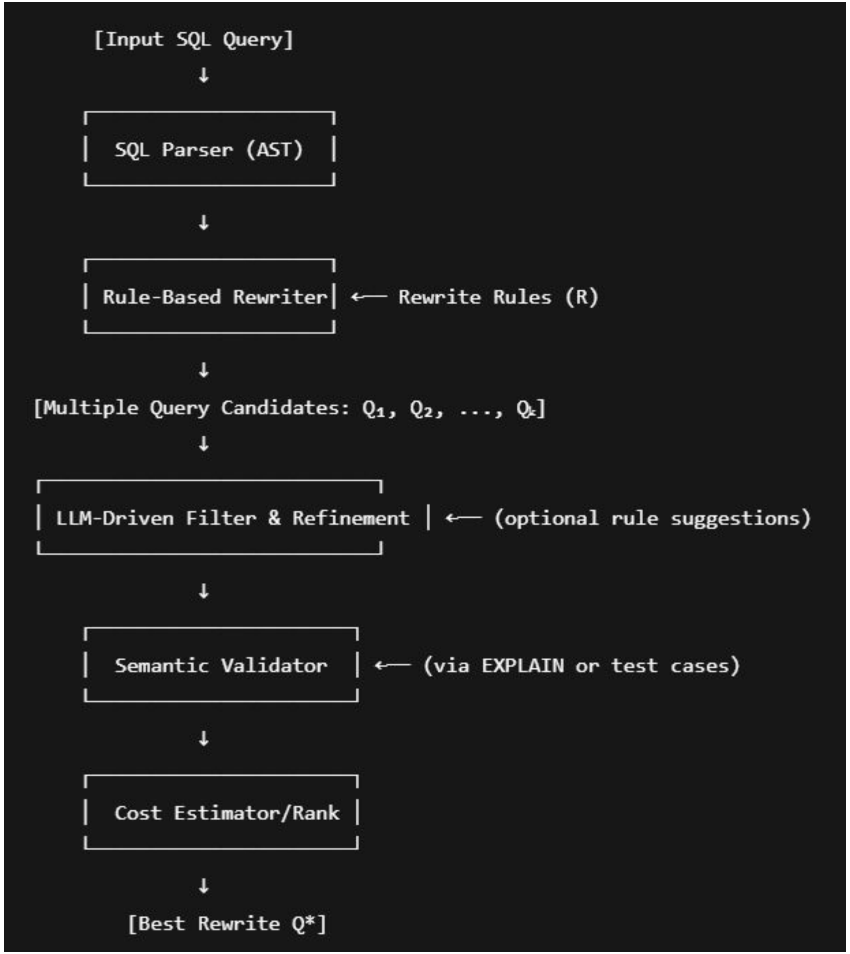
Combining Multi-Query Generation with Rule-Based Optimisation

Key Concepts

- **Query rewriting:** Transforming a SQL query into an equivalent one that may be more performant.
- **Rule-based optimisation:** Using deterministic, algebraic transformation rules to generate logically equivalent rewrites.
- **Rewrite sequence search:** Exploring multiple rule application sequence.
- **LLM-driven filtering:** Using a language models to score, rank, or suggest improved rewrites, or to select which rules to apply next.
- **Semantic equivalence:** Ensuring transformed queries return the same results as the original query.
- **Performance awareness:** Incorporating cost estimating into rewrite selection.

Problem Formulation

- **Goal:** Given an input SQL query Q, generate and select a semantically equivalent query Q' such that Q' improves performance using a combination of rule-based transformation and LLM-based guidance.



Rule-based Rewriter Module

- **Goal:** Generate multiple equivalent rewrites of a SQL query using a predefined set of rules.
- Subcomponents:
  - SQL to Logical Plan converter
  - Rule application engine
  - Search strategy:
    - Greedy: Apply one rule at a time
    - Beam Search: Keep top-k candidates at each depth
    - MCTS: Use a tree policy and rollout evaluation

LLM-Driven Rewriting and Filtering

- **Goal:** Use an LLM to refine or filter candidates based on semantic correctness or performance intuition.
- **Subtasks:**
    - **Rewrite ranking:** Ask the LLM to compare Q_1, Q_2, ..., $Q_{\square}$ and rank based on clarity, expected performance, or simplicity.
    - **Next rule suggestion:** Ask the LLM what transformation would improve Q_i.
    - **Rule selection justification:** Prompt the LLM to explain why a specific rewrite is valid and useful (helps with transparency).

Semantic Equivalence Validator

- **Goal:** Ensure rewritten queries are equivalent to the original.
- **Subtasks:**
    - **Test-query evaluation:** Run queries on sample data and compare outputs.
    - **Result signature comparison:** Use schema-based output matching and result hashes.
    - **Counterexample generation**: Create rows that differentiate queries.

Cost Estimation and Ranking

- **Goal:** Score and rank candidates based on performance.
- **Subtasks:**
    - Use database-native EXPLAIN (ANALYZE) to get estimated/actual cost.
    - Optional: Train a simple cost model (e.g., decision tree on query features).
    - Normalize and rank queries by cost.

References

1. https://www.vldb.org/pvldb/vol16/p4110-li.pdf
2. https://www.vldb.org/pvldb/vol18/p53-yuan.pdf
3. https://arxiv.org/html/2403.09060v1#:~:text=smarter%20and%20more%20effective%20over,the
4. https://aclanthology.org/2023.emnlp-main.322.pdf
5.

Plan

- Prepare dataset
- Traditional Query Optimisation Techniques
- LLM-based Query Optimisation Techniques
- Ensemble method
- Cost estimation for query
-