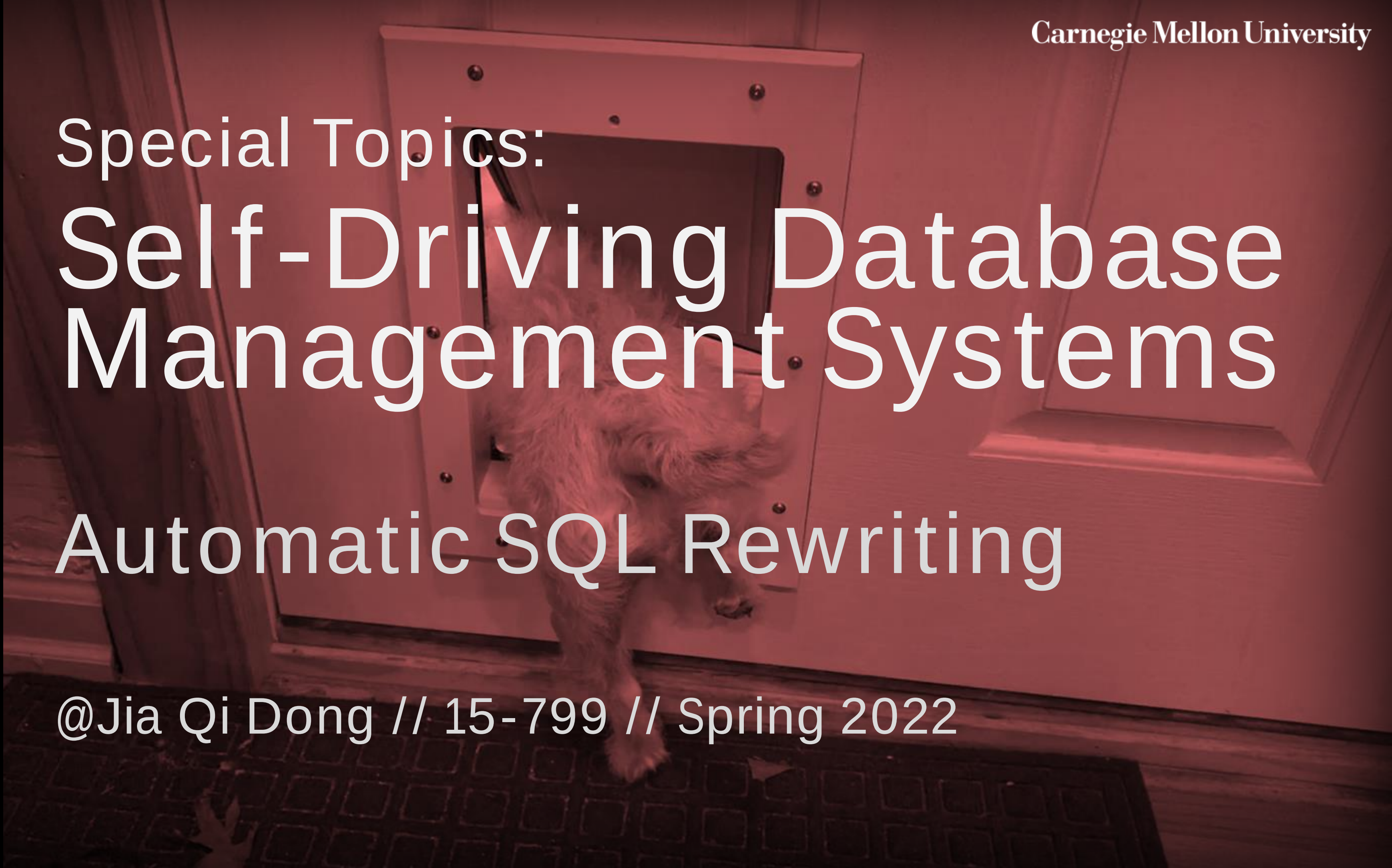


Lecture #16

Carnegie Mellon University

Special Topics:

Self-Driving Database Management Systems

Automatic SQL Rewriting

@Jia Qi Dong // 15-799 // Spring 2022

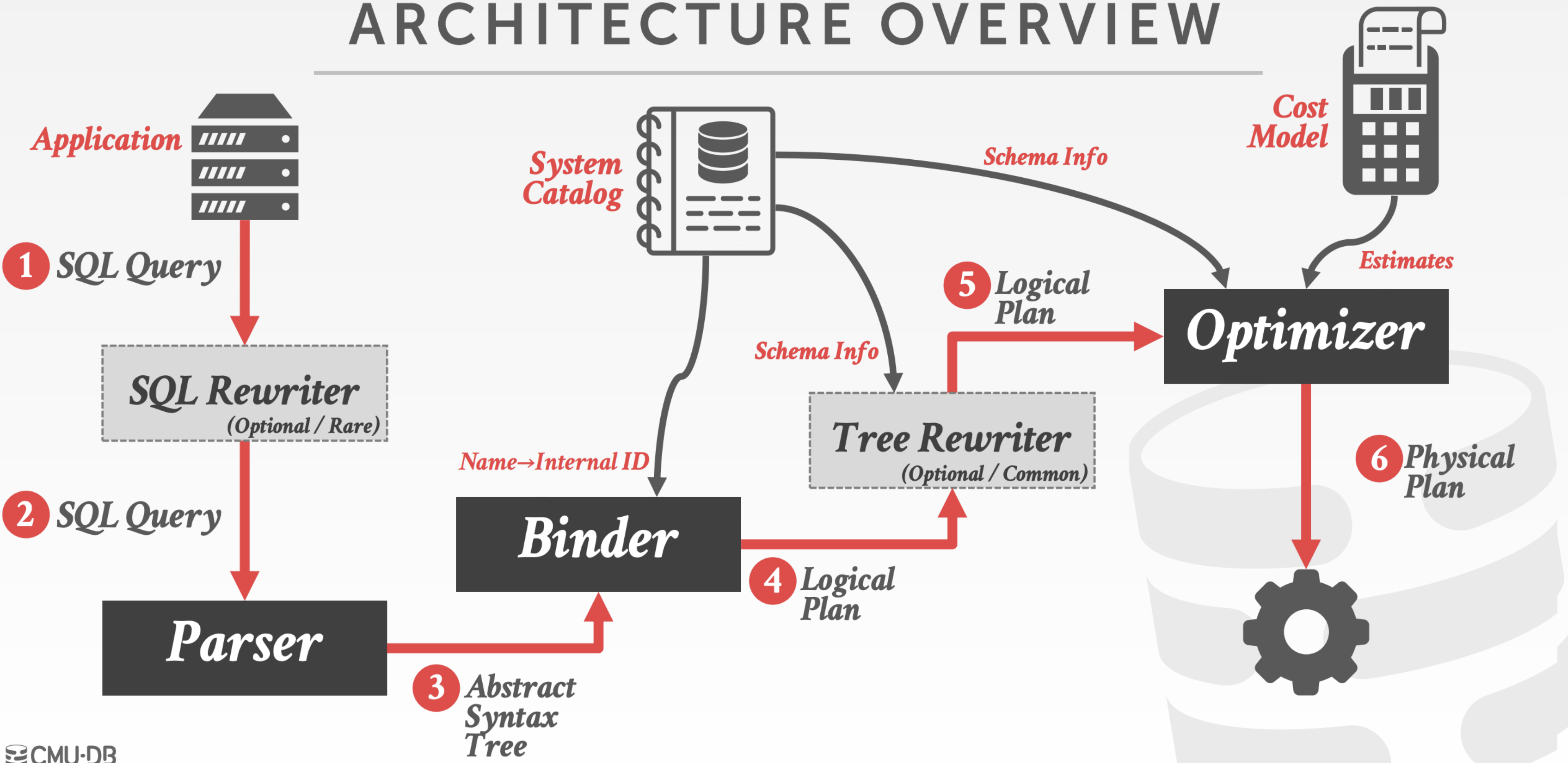
LAST CLASS (DATAFARM)

- Generate abstract plans through Markov Chains
- Instantiate them based on real workload's distribution

TODAY'S AGENDA

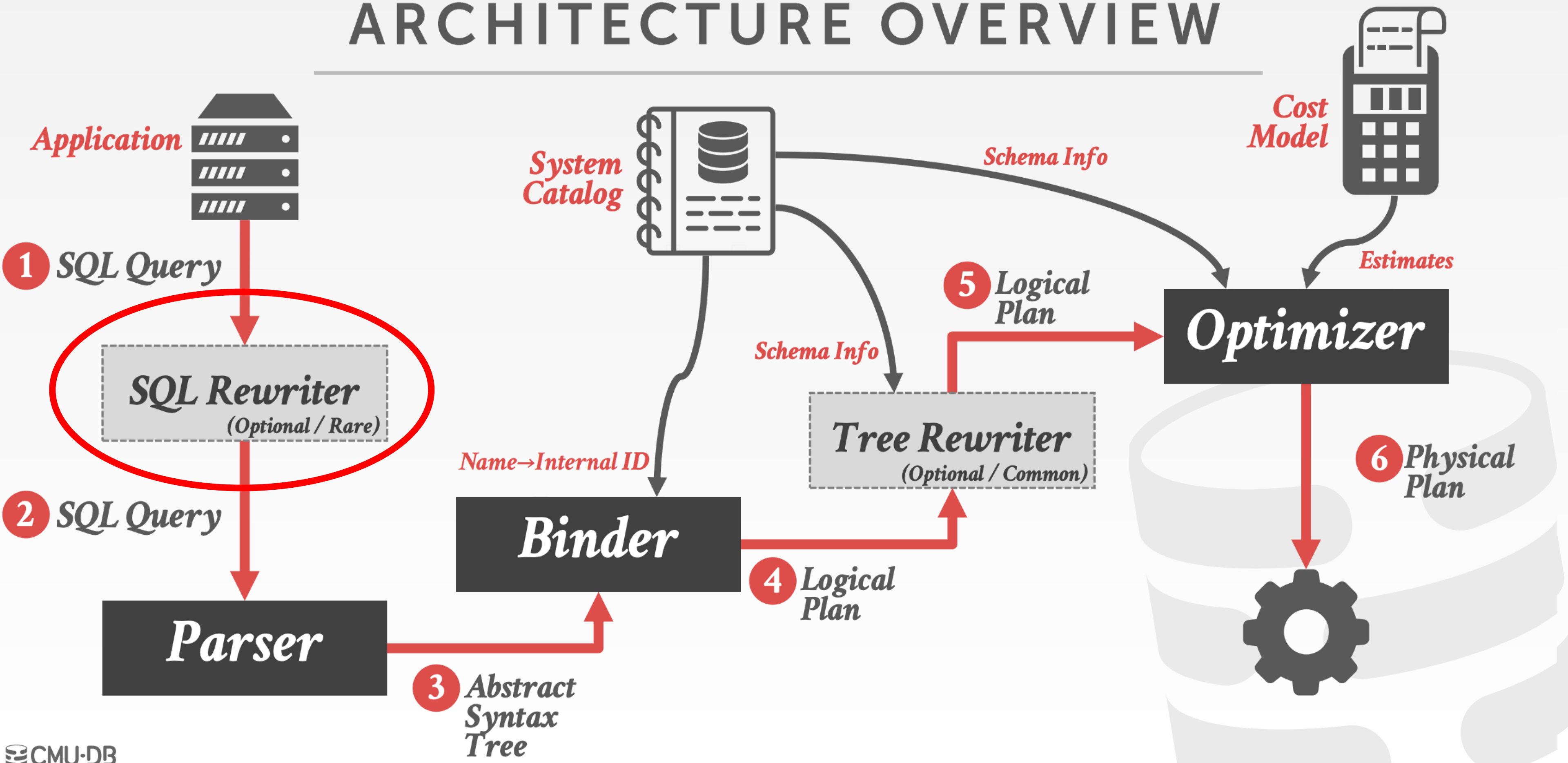
- Overview
- MCTS
- Deep Rewrite Estimation
- Experiments

ARCHITECTURE OVERVIEW



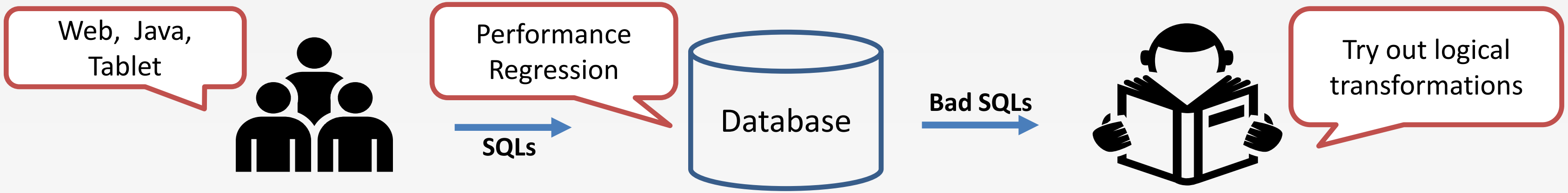
SOURCE: CMU 15645 – QUERY PLANNING PART 1

ARCHITECTURE OVERVIEW



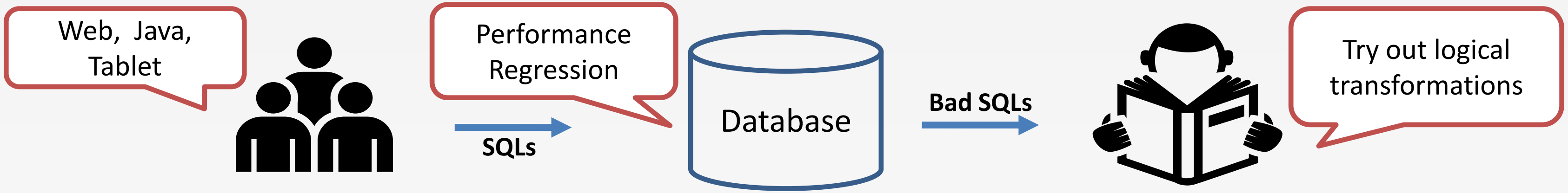
SOURCE: CMU 15645 – QUERY PLANNING PART 1

MOTIVATION



SOURCE: XUANHE ZHOU

MOTIVATION




```
SELECT MAX(DISTINCT L1.col1)
FROM lineitem L1
WHERE L1.col1 = ANY (
            SELECT MAX(C.col1) m_key
            FROM customer C, lineitem L2
            WHERE C.col1 = L2.col1
              AND  ((C.col2<2 AND
C.col3<2)
OR(C.col2<2 AND L2.col2>5))
            GROUP BY C.col1
);
```

Execution Time > 20min

Over 600x ↑

```
SELECT MAX(L1.col1)
FROM lineitem L1, customer C
WHERE L1.col1 = C.col1



  AND  (C.col2<2 OR (C.col3<2 AND
L2.col2>5))
;
```

Execution Time > 1.941s

SOURCE: XUANHE ZHOU

REWRITE RULES

- Remove redundant aggregates

```
select max(distinct a) from t; →
select max(a) from t;
```

- Create temporary table

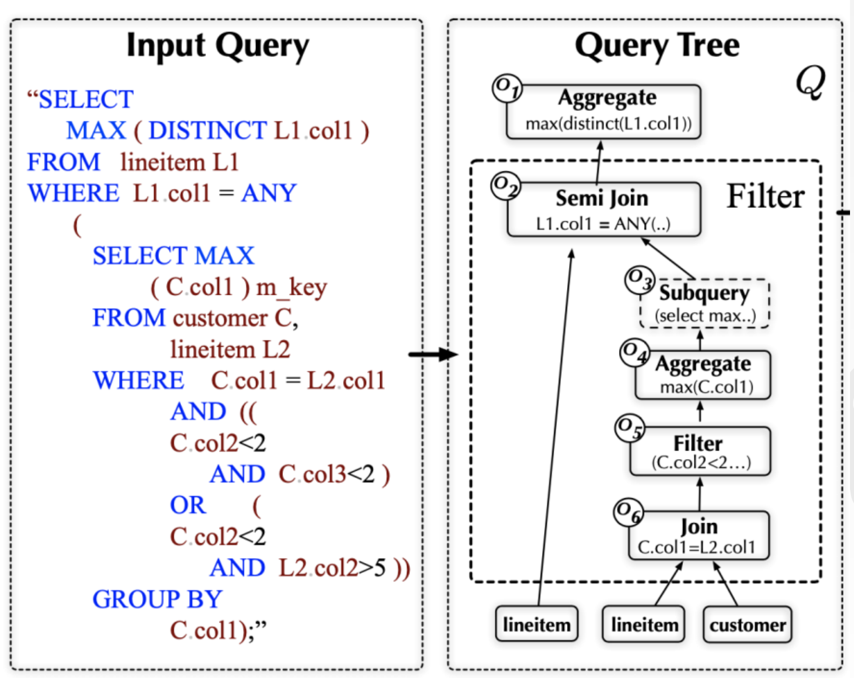
```
select * from t1 where a1 <  any(select a2 from t2);
→
With t as (select a2 a from t2)
select t1.* from t1, t where a1<a
```

EXISTING APPROACHES

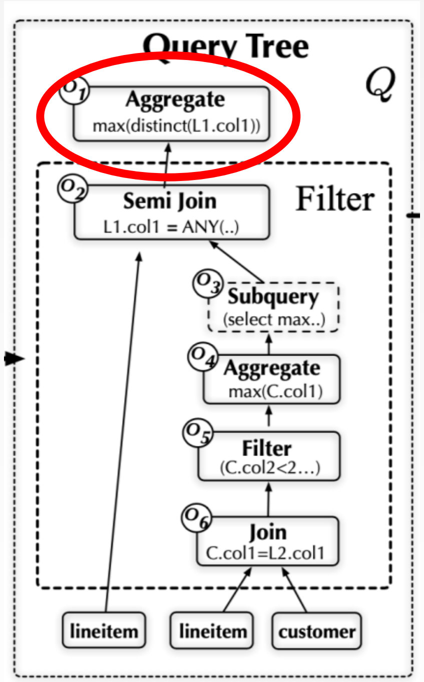
- DBAs rewrite queries based on some rules and past experiences
  - Problem: can't generalize to a large number of queries
  - Can't decide which rules are better

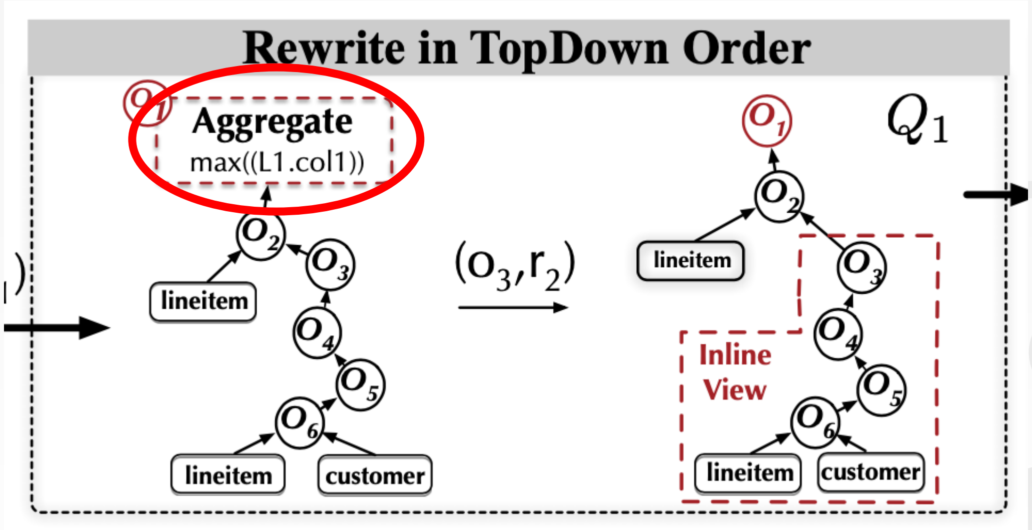
- Heuristic query rewrite (Postgres)
  - Problem: may find local optimum instead of global optimum

REWRITE ORDER

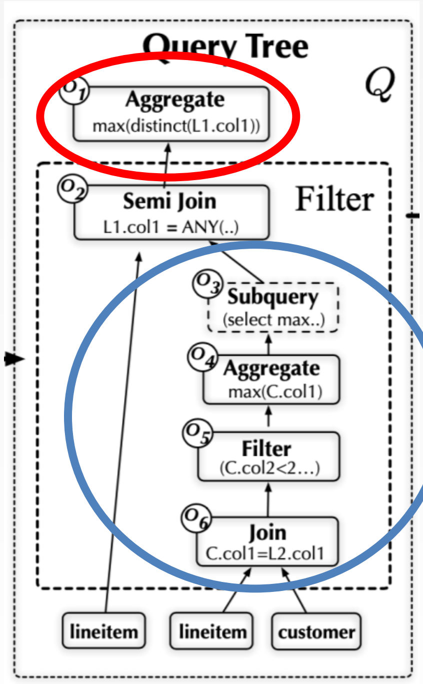


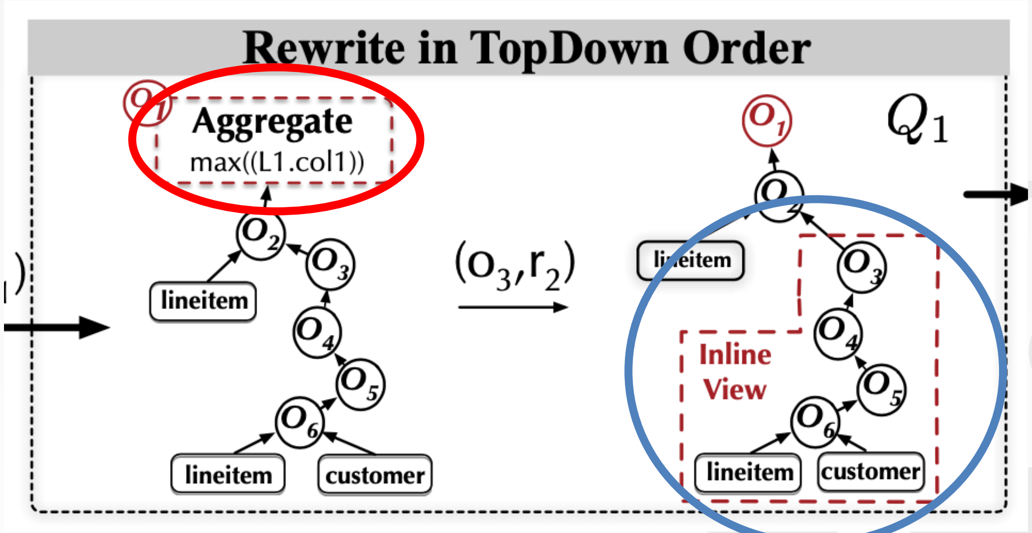
TOP-DOWN ORDER (POSTGRES)



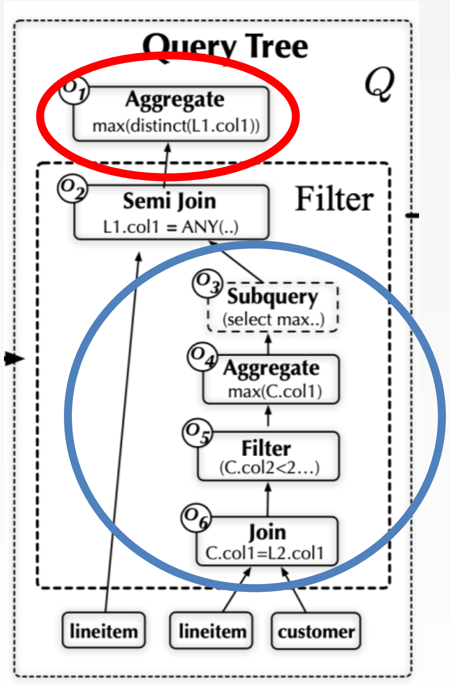


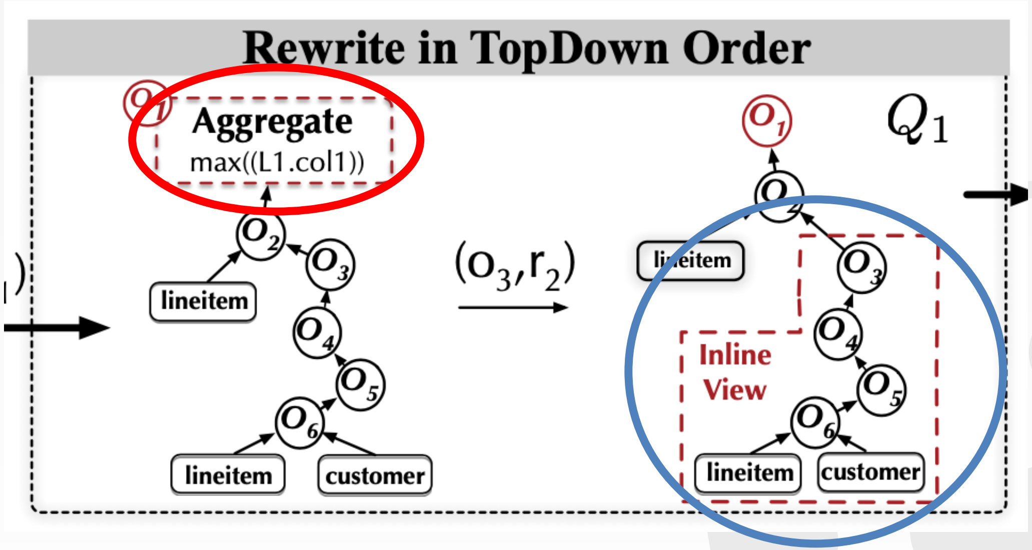
TOP-DOWN ORDER (POSTGRES)





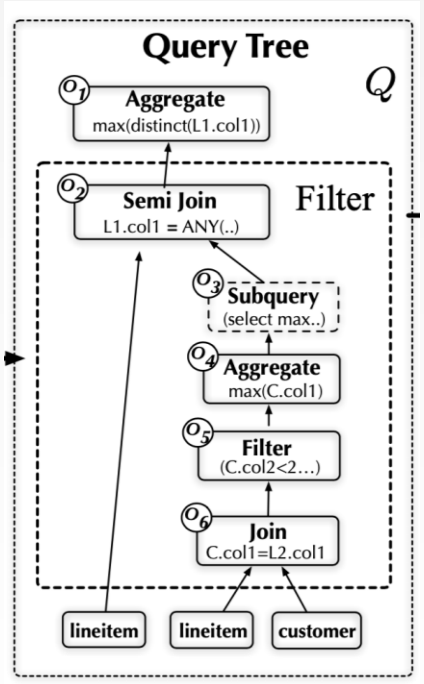
TOP-DOWN ORDER (POSTGRES)

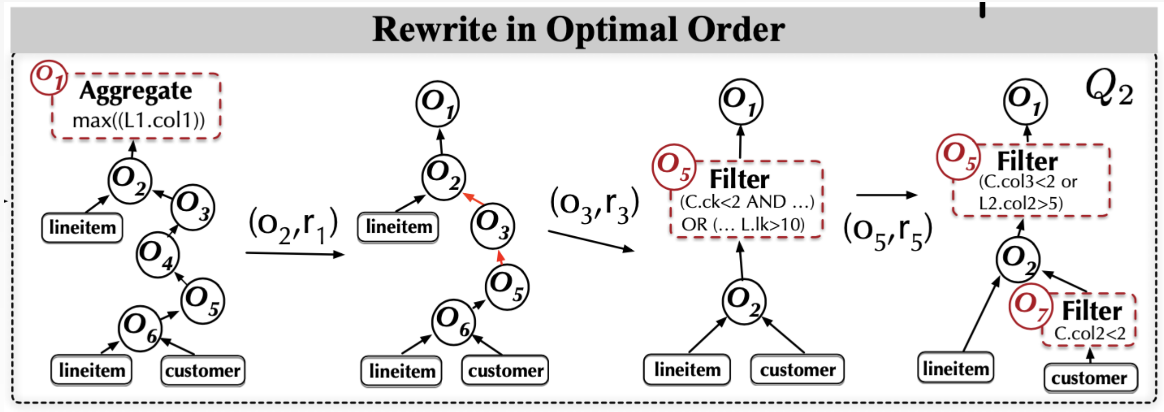




Cannot further optimize temporary table
Execution: > 20 min

OPTIMAL ORDER





OPTIMAL ORDER

Input Query

```
"SELECT
    MAX ( DISTINCT L1.col1 )
FROM   lineitem L1
WHERE  L1.col1 = ANY
    (
      SELECT MAX
          ( C.col1 ) m_key
      FROM customer C,
             lineitem L2
      WHERE    C.col1 = L2.col1
             AND  ((
             C.col2<2
                 AND  C.col3<2 )
             OR      (
             C.col2<2
                 AND  L2.col2>5 ))
      GROUP BY
             C.col1);"
```

Rewritten Query

```
SELECT MAX(L1.col1)
FROM lineitem L1, customer C
WHERE L1.col1 = C.col1
   AND  (C.col2<2 OR (C.col3<2 AND
L2.col2>5));
```

Execution: 1.941 s

CHALLENGES

- **Goal:** Transform a SQL query into an **equivalent** query with **minimal execution time**
- Challenges:
  - How to represent large amount of possible ordering?
    - 10 rules → 30,000 different rewritten queries
  - How to find optimal ordering of rules efficiently?
    - Rewrite requires low overhead (milliseconds)
    - Need a light-weight model/algo
  - How to estimate benefit of whole rewrite order?
    - Rewrite one operator may affect benefit of others

TODAY'S AGENDA

Overview

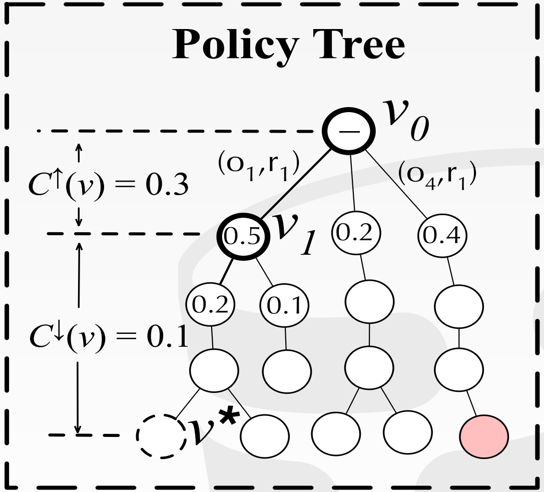
MCTS

Deep Rewrite Estimation

Experiments

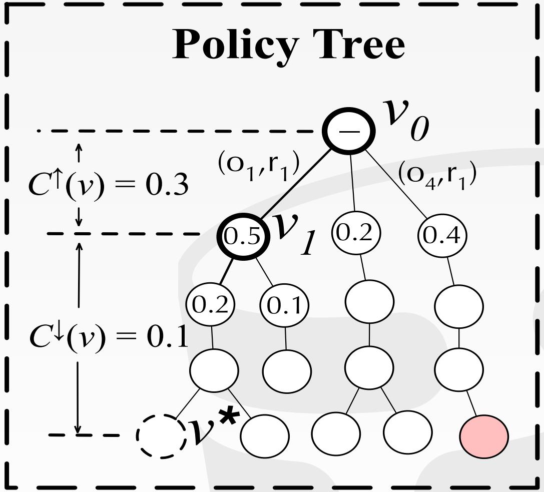
MONTE CARLO TREE SEARCH

- RL algorithm commonly used in tree search
  - AlphaGo
- Root = original query
- Child node is obtained by **applying a rewrite rule to an operator**



MONTE CARLO TREE SEARCH

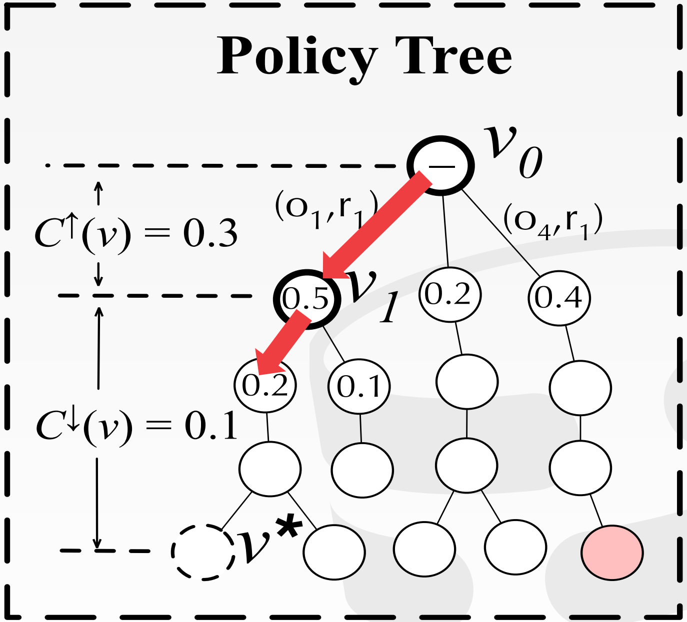
- Node → a rewritten query
- Edge → (operator, rule) pair
- C↑ → cost reduction from v_0
  - Cost estimated by optimizer
  - $C^{\uparrow}(v_i) = \mathrm{Cost}(v_0) - \mathrm{Cost}(v_i)$
- C↓ → Subsequent cost
  - Estimated using NN
  - $C^{\downarrow}(v_i) = \mathrm{Cost}(v_i) - \mathrm{Cost}(v_i^*)$
- F → Access frequency



$$\mathcal{U}(v_i) = (C^{\uparrow}(v_i) + C^{\downarrow}(v_i)) + \gamma\sqrt{\frac{ln(\mathcal{F}(v_0))}{\mathcal{F}(v_i)}}$$

MONTE CARLO TREE SEARCH

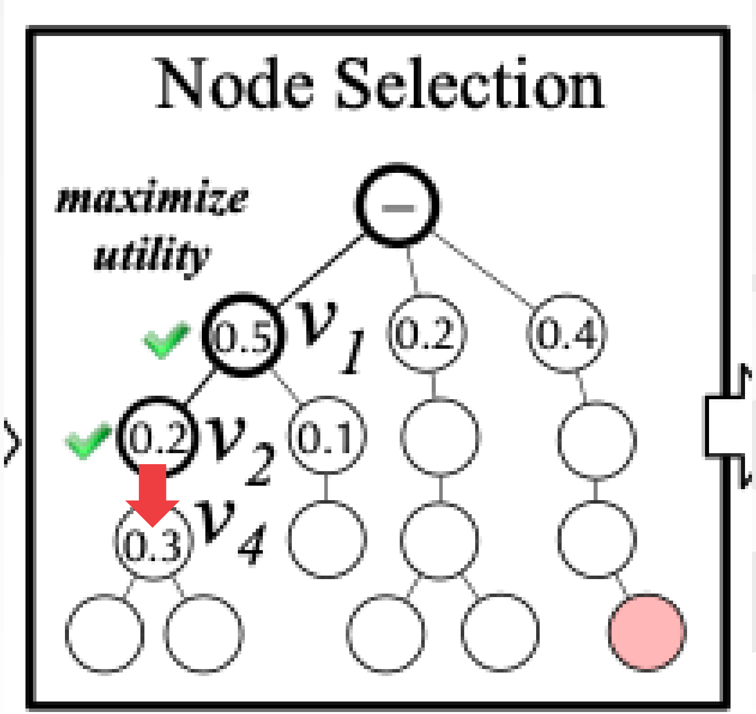
1. Selection
   → V2 is selected



MONTE CARLO TREE SEARCH

1. Selection

2. Expansion
   For each operator o:
      For each rule r:
         if (o,r) appliable:
            add child node

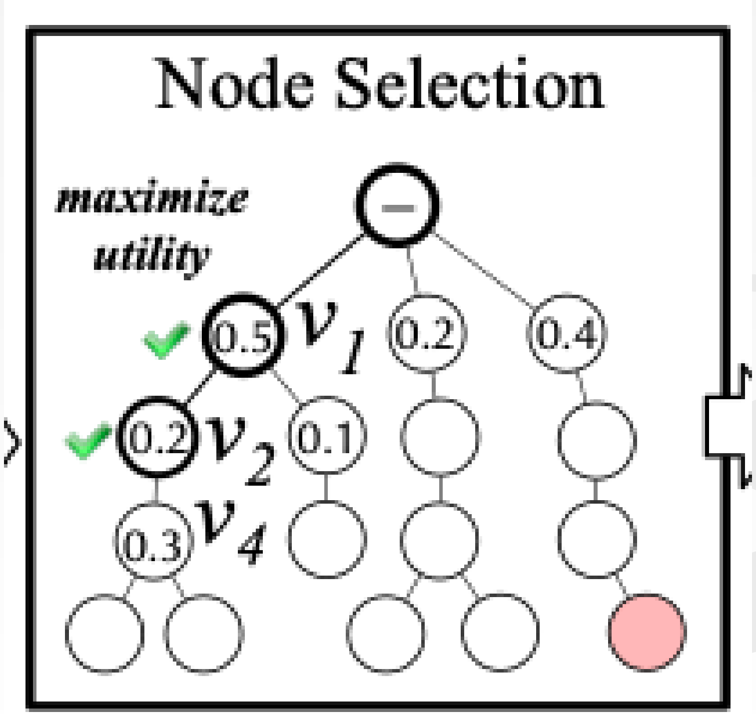


MONTE CARLO TREE SEARCH

1. Selection

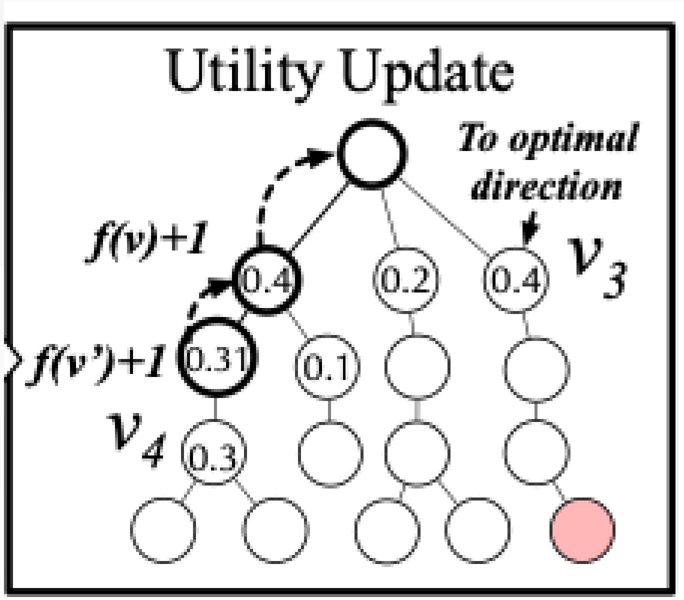
2. Expansion

3. C↓ Estimation
   → C↓(v2) = 0.1



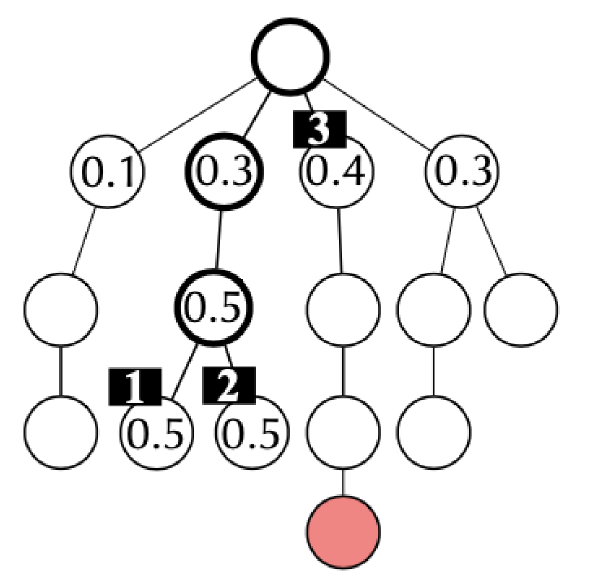
MONTE CARLO TREE SEARCH

1. Selection
2. Expansion
3. C↓ Estimation
4. Utility Update
   - → If ancestor v' has smaller benefit, update C↓(v')
   - → Increase F(v')
   - → Update U(v')



PARALLEL QUERY REWRITE

- Select multiple nodes to expand at each iteration
- The nodes do NOT have ancestor-descendant relationships
- Sum of their utilities is maximized



- Dynamic Programming !

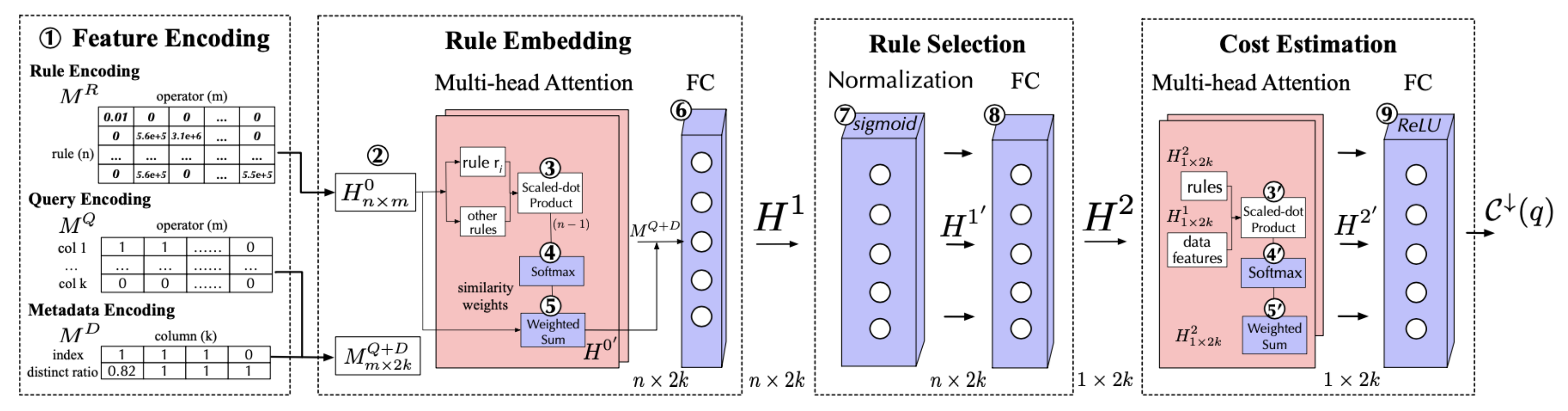
TODAY'S AGENDA

Overview
MCTS
Deep Rewrite Estimation
Experiments

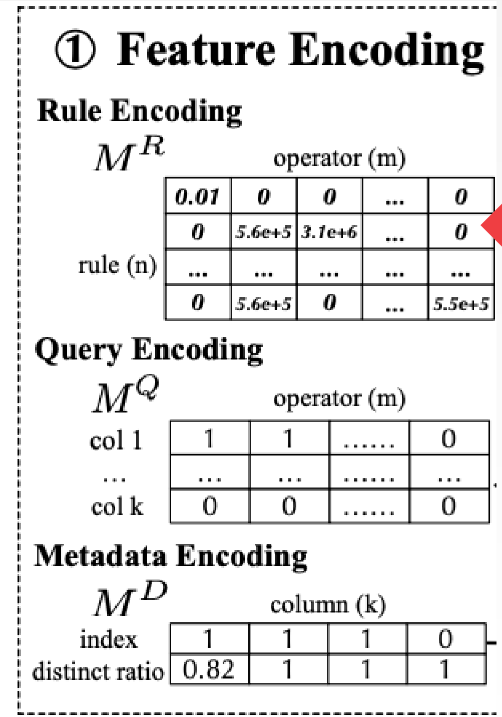
CHALLENGES

- Goal → Estimate C↓
- Challenges
  - Many factors affecting cost reduction (query operator, rules, data distribution …)
  - Rewrite rules are correlated (applying a rule make other rules unapplicable)
  - Expensive to obtain labelled training data

MULTI-HEAD ATTENTION MODEL

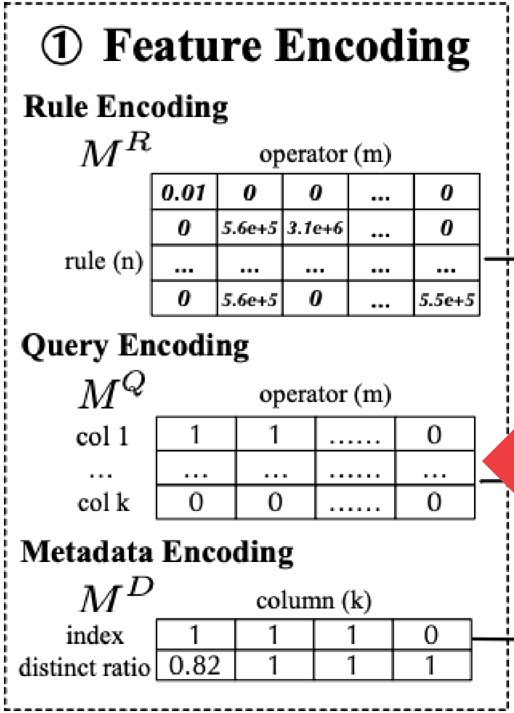


RULE ENCODING



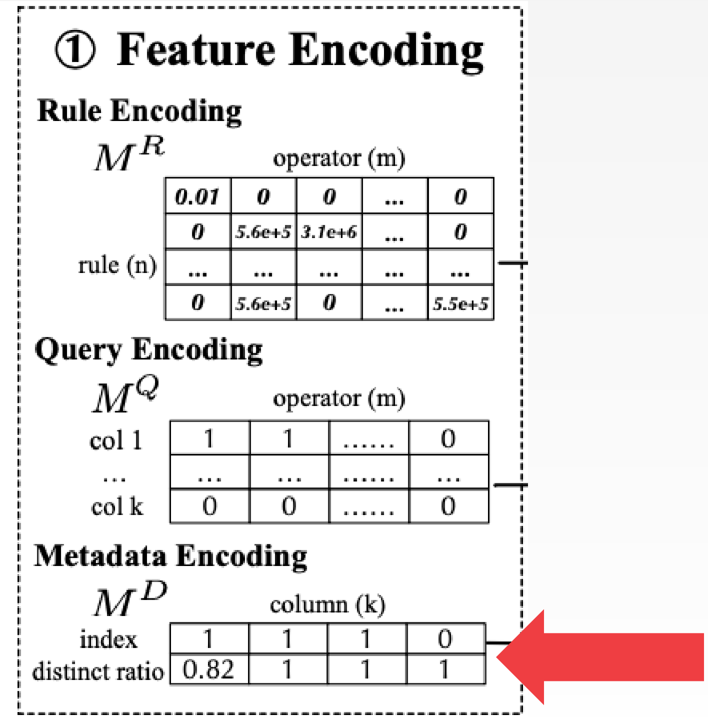
- What is the cost of each (operator, rule) pair?
- **M^R[i, j]** = cost reduction of applying rule i on operator j
- **M^R[i, j] = 0** if not applicable
- Estimated by the optimizer

QUERY ENCODING



- What columns does each operator reference?
- **M^Q[i, j] = 1** if operator j contains column i
- **M^Q[i, j] = 0** otherwise

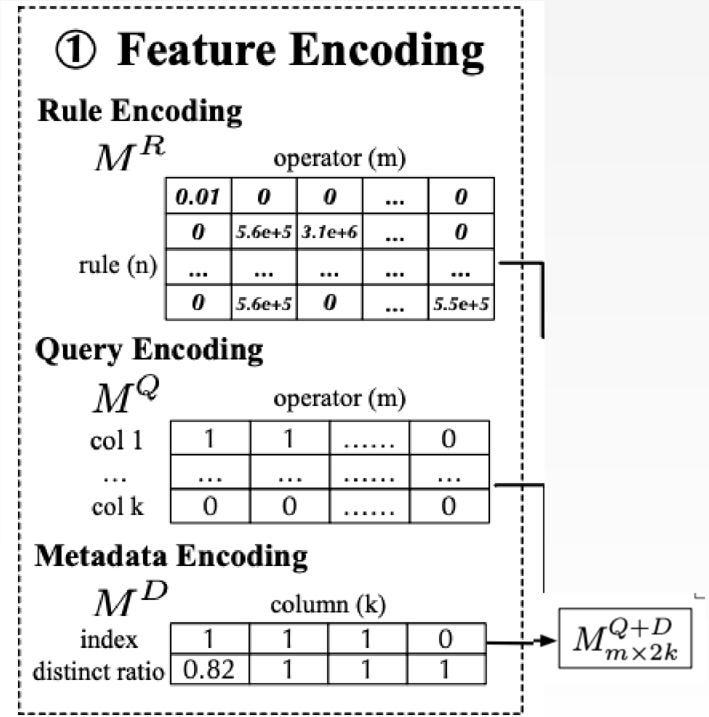
METADATA ENCODING



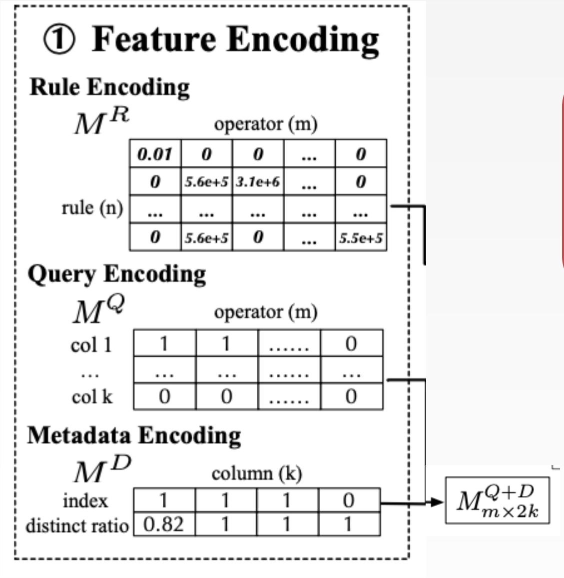
- Does column i have index?
- $M^D[0, j] = 1$ if column j has index

- How many distinct values does each column have?
- $M^D[1, j]$ = distinct value ratio of column j

COMBINED ENCODING

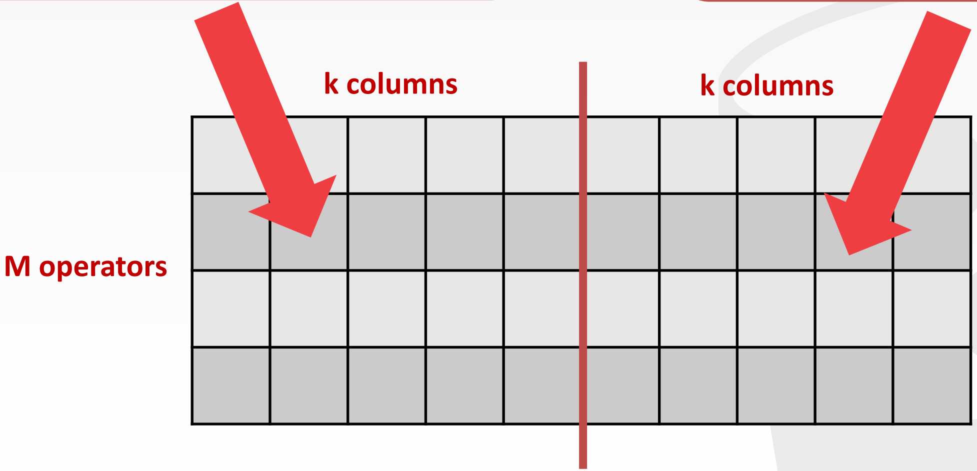


COMBINED ENCODING

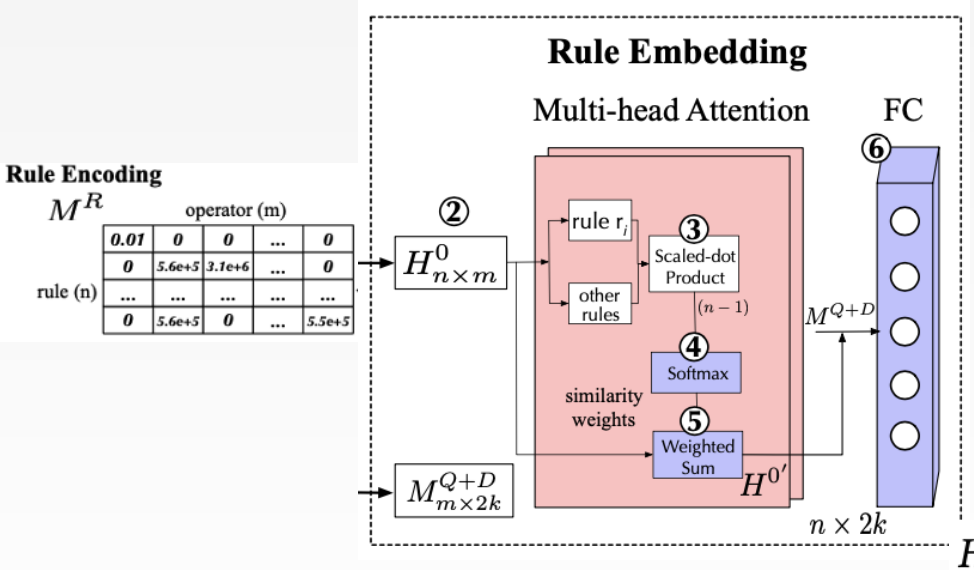


- Does this operator have column j, and does column j have index?
- **1** if both conditions are true

- Does this operator have column j, and what is its distinct ratio?



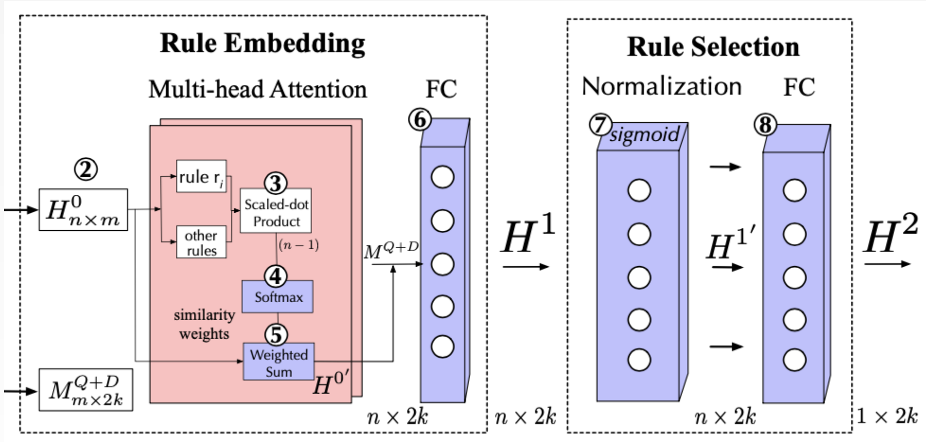
RULE EMBEDDING



- Attention = Similarity
- Compute similarity of each rule (row) wrt other rules (other rows)
- Similar rules
  - affect the same operators by the same amount
  - Likely to cause rewrite conflict

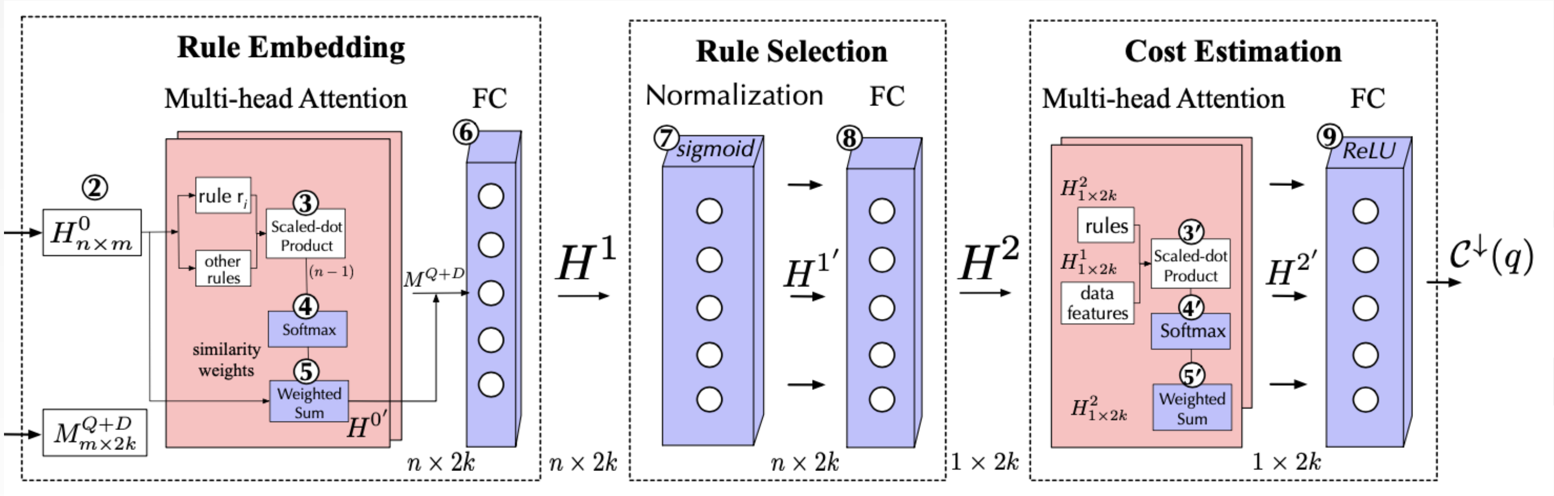
$$H^{0'}[i,*] = H^0[i,*] + \sum_{j\neq i} \frac{1}{\alpha_{i,j}} H^0[j,*]$$

RULE SELECTION



- H^1 has n rows → rewrite benefits of all rules
- H^2 has 1 row → represents rule with optimal benefits

COST ESTIMATION



TRAINING DATA GENERATION

- Randomly assemble {Table, Join, Predicate, Aggregate Operation, Column}
- Use SQLSmith to synthesize queries via SQL syntax tree
- Cluster queries (DBSCAN) based on cost vectors
- Sample 5% queries to enumerate optimal rewrite costs
- Take avg cost reduction as label for queries in the cluster

LOSS FUNCTION

- Many noises in training data since labels are based on small part of queries
- $L(q) = \left(F(q) - C \downarrow (q)\right)^2 + \sum_{i,j} \mu_{i,j} |F(qi) - F(qj)|$
- Minimize MSE between estimation model and labelled cost reduction
- Minimize L1 distance between queries in the same cluster as they should have similar cost reduction

TODAY'S AGENDA

Overview
MCTS
Deep Rewrite Estimation
Experiments

EXPERIMENT SETTING

- **Machine**: 16GB RAM, 256GB disk, 4.00GHz CPU, Titan RTX 2080Ti GPU with 11 GB buffer
- **Dataset**:
  - TPC-H 1x (~4.7G), TPC-H50x (~50G)
  - JOB (~1.1G) → **OLAP** with IMDB data, 16,000 queries
  - XuetangX (~11.5G) → **OLTP** benchmark, 22,000 queries
- **Metrics**
  - Execution cost from query optimizer
  - Rewrite latency: time of rewriting a query
  - Query Latency: time of answering a query
  - Overall Query Latency: rewrite latency + query latency

EXECUTION COST AND QUERY LATENCY

- Baseline
  - TopdownPostgres
  - TopdownCalcite
  - Heuristic → Always select the rule with most benefit
  - Arbitrary → randomly select operators

- Tree Search + cost estimation = better rewrite orders

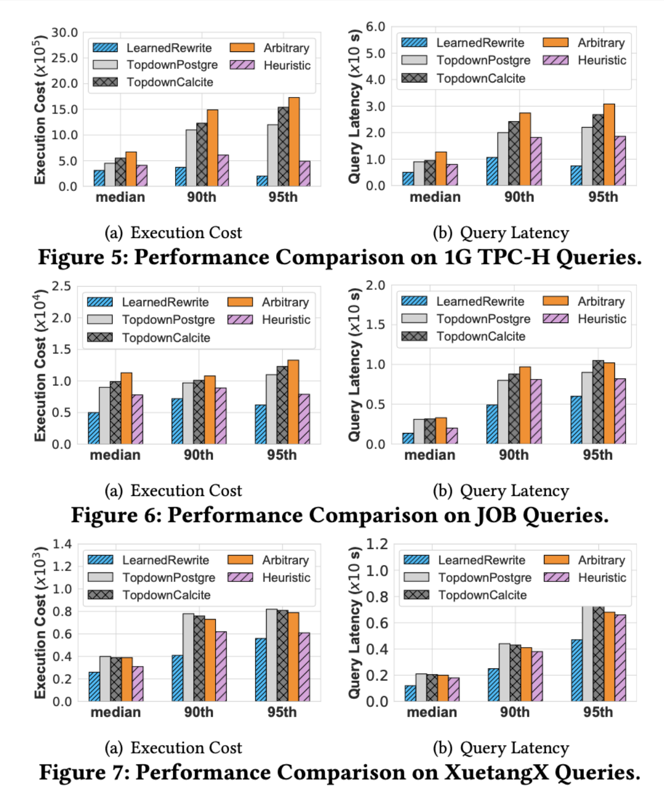


Figure 5: Performance Comparison on 1G TPC-H Queries.

Figure 6: Performance Comparison on JOB Queries.

Figure 7: Performance Comparison on XuetangX Queries.

REWRITE LATENCY

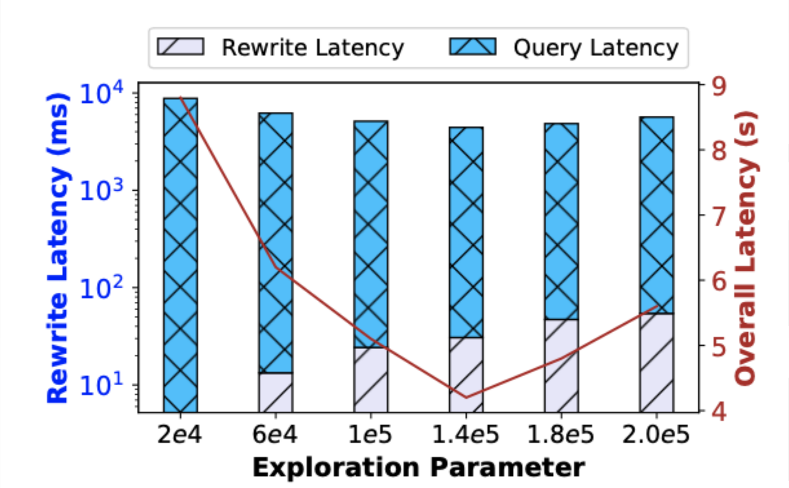
- LearnedRewrite has high rewrite latency, but lowest overall latency

Table 4: Average Rewrite/Query Latency on 50G TPC-H.

Method	Rewrite Latency	Query Latency
Arbitrary	3.3 - 10.1 ms	553.2 s
TopdownPostgre	**0.3 - 3.9 ms**	427.5 s
TopdownCalcite	1.5 - 18.9 ms	431.1 s
Heuristic	5.8 - 24.2 ms	331.7 s
LearnedRewrite	6.1 - 69.8 ms	**224.5 s**

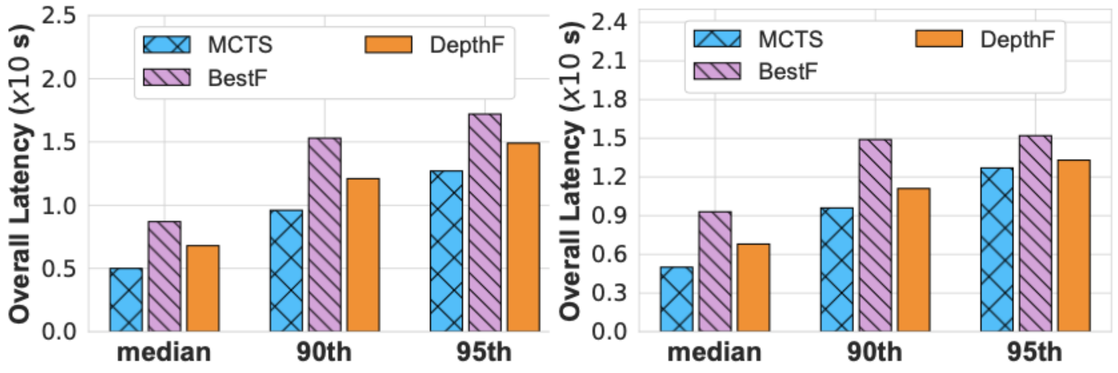
EXPLORATION PARAM

- ↑ γ →↑rewrite latency; ↓query latency
- Sweet spot at $\gamma = 1.4 \times 10^5$
- Use higher γ for slow queries
  - Main bottleneck is query latency
- lower γ for fast queries
  - Reduce rewrite overhead



TREE SEARCH ALGO

- Baseline: DFS, B(est)FS

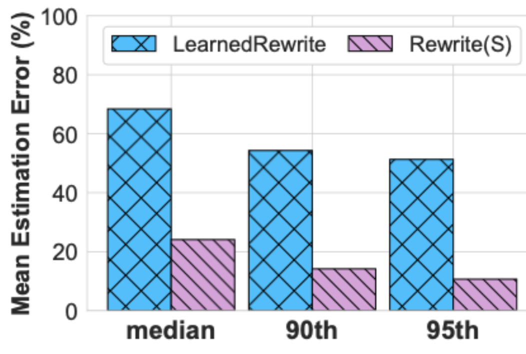


(a) Rewrite Latency (b) Overall Latency

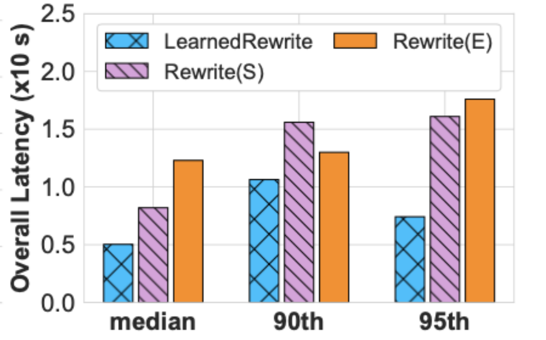
Figure 11: Comparison of Tree Search Algorithms (TPC-H).

ESTIMATION MODEL

- Tested against 2 strategies:
  - Rewrite(E): Enumerate all rewrite orders
  - Rewrite(S): Sample K rewrite orders
- LearnedRewrite produces faster queries

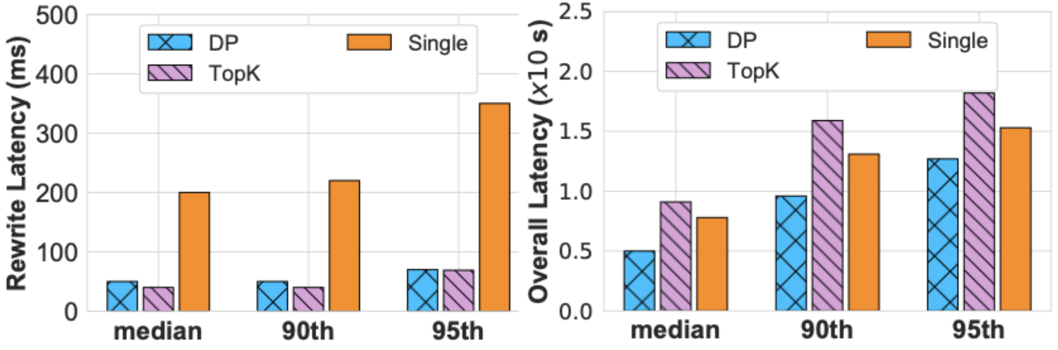


(a) Estimation Accuracy



(b) Overall Latency

PARALLEL REWRITE



(a) Rewrite Latency (b) Overall Latency

Figure 10: Comparison of Parallel Algorithms (TPC-H). DP denotes DP-based parallel rewrite; TopK denotes greedy parallel rewrite; and Single denotes selecting single node.

VARYING NUMBER OF REWRITE RULES

- LearnedRewrite's model can adapt to different rule combinations (pad unused rule as 0)
- Can also efficiently search in large search space

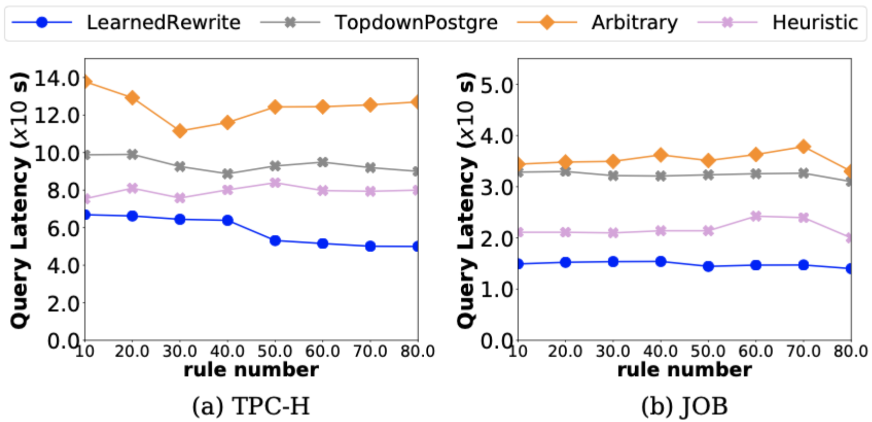
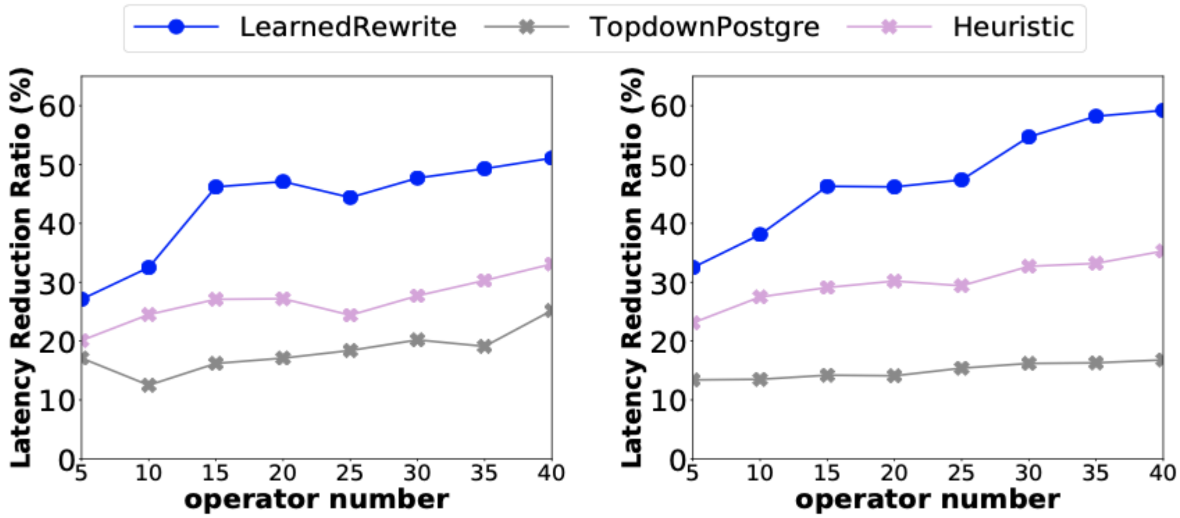


Figure 12: Adaptability on Different Number of Rules.

VARYING NUMBER OF OPERATORS



(a) TPC-H (b) JOB

PRBOLEMS

- Retrain if new tables/columns
- Similarity = rewrite conflict?
- All queries in the cluster have the same label?

- Demo