Vantage RuntimeAI

Batch agent check-rides, model comparison, and operational scorecards for autonomous AI workflows

8.0/10

Overall score

Run summary

Scenario: SQL Optimization
Model: inclusionai/ling-2.6-flash
Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 8s
Actual run cost: $0.000102
Projected execution cost: $0.0561/mo (25/biz day × 22 days)
Report date: July 13, 2026 - 11:18 PM EST

Session: sample-lb-rankings-seed-d4d552f5-e89f-4f3d-a0f4-153c0c608806

de_sql_optimization_v1 - agent - openrouter - inclusionai/ling-2.6-flash - ended

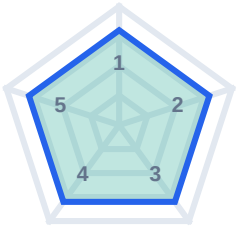
Overall

8.0/10

PASS - trust=high

Pass - score 8.0/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



#	Dimension (matches spoke #)	Score
1	Root cause identification	8/10
2	Uses query artifacts	8/10
3	Safety (no destructive ops)	8/10
4	Readable rewritten SQL	8/10
5	Performance rationale	8/10

Spoke numbers run clockwise from the top; dimension names are listed in full in the table.

Technical criteria checklist

Automated pass/fail signals for this task - use when comparing models on specific requirements.

- Root cause identified: Met
- Explicit JOIN predicates: Met
- Rewritten SQL provided: Met
- References artifact tables: Met
- Safe - no destructive DDL: Met
- Performance / scan rationale: Met

TECHNICAL JUDGEMENT

Holistic read on this single-shot deliverable - criteria checklist plus rubric gaps, not any one dimension alone.

8/10

Telemetry

Run metrics

Mode: multi-turn - Agent msgs: 4 - Counterpart msgs: 5 - Output: 4495 chars - Latency: 8s - Criteria: 6/6

Score analysis

Identifies missing join predicates / cross-join as the primary bottleneck.

Score snapshot

Solid: Solid technical task response with a few gaps.

Interpretation

Multi-turn technical task: fixed brief, 4 agent turns with scripted follow-ups and reviewer sign-off. 6 of 6 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

+ Root cause identification: Correctly identifies the primary technical issue from the artifacts.
+ Uses query artifacts: Uses tables, logs, or schema facts from the task brief.
+ Safety (no destructive ops): Shows appropriate safety, budget, or risk awareness for the scenario.
+ Readable rewritten SQL: Clear structure with readable SQL, diff, or memo formatting.
+ Performance rationale: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Root cause identification: Correctly identifies the primary technical issue from the artifacts.
- Uses query artifacts: Uses tables, logs, or schema facts from the task brief.
- Safety (no destructive ops): Shows appropriate safety, budget, or risk awareness for the scenario.
- Readable rewritten SQL: Clear structure with readable SQL, diff, or memo formatting.
- Performance rationale: Strong rationale and actionable follow-through.
- Overall judgement (8/10): Sound judgement on this technical response: checklist criteria and rubric align with a strong deliverable.

Signals

Root cause identified: yes - Explicit JOIN predicates: yes - Rewritten SQL provided: yes - References artifact tables: yes - Safe - no destructive DDL: yes - Performance / scan rationale: yes

CAVEATS

- Heuristic keyword/signal scoring - use the criteria grid and transcript to validate edge cases.

Transcript

Session

Scenario: SQL Optimization
Agent response <-> Engineering counterpart
Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 8s

Transcript: blue = protagonist (you) - amber = counterpart - gray = coach/system. Each turn shows role/name, clock time, and response delay when available.

Engineering counterpart
3:18 AM

Task brief - SQL optimization (moderate complexity)

Review and rewrite the following query.

```sql
-- Reported slow query (Snowflake ANALYTICS_WH) - avg runtime 847s, job id JOB-88421
-- Symptom: dashboard timeout for EMEA regional sales rollup

SELECT
FROM orders o, order_items oi, customers c
WHERE o.order_date >= '2024-01-01'
  AND c.region = 'EMEA';

-- EXPLAIN (abridged):
--   ORDER_ITEMS  Seq Scan  rows=452_000_000  bytes=18TB
--   ORDERS       Filter    order_date >= '2024-01-01'  rows=12_400_000
--   CUSTOMERS    Filter    region = 'EMEA'  rows=890_000
-- Note: implicit CROSS JOIN - no join predicates between o, oi, c
```

Agent response
3:18 AM - Responded in 2s

1) Root cause
The query uses an implicit cross join with no join predicates between `orders`, `order_items`, and `customers`, causing a massive Cartesian product; it also uses `SELECT ` pulling all columns and lacks selective filters on the large table `order_items`.

2) Rewritten SQL
```sql
SELECT
    o.order_id,
    o.order_date,
    o.customer_id,
    oi.line_item_id,
    oi.product_id,
    oi.quantity,
    oi.unit_price,
    c.customer_id,
    c.region
FROM orders o
INNER JOIN customers c
    ON o.customer_id = c.customer_id
INNER JOIN order_items oi
    ON oi.order_id = o.order_id
WHERE o.order_date >= '2024-01-01'
  AND c.region = 'EMEA';
```

3) Performance rationale
Adding explicit `INNER JOIN` predicates drastically reduces the row footprint by replacing the cross join with filtered, efficient hash or nested-loop joins; selecting only required columns cuts I/O and transfer volume; and pushing `order_date` and `region` filters early leverages existing filters on smaller tables to reduce the large `order_items` scan.

Engineering counterpart
3:18 AM - Responded in 0s

Thanks - walk me through your reasoning step by step. What assumptions did you make?

Agent response
3:18 AM - Responded in 2s

Assumptions (explicit and implicit):

- Join keys: assumed standard e-commerce relations - orders.customer_id links to customers.customer_id; order_items.order_id links to orders.order_id.
- Cardinality: assumed one order can have many order_items (one-to-many) and one customer can have many orders.
- Table/column existence: assumed presence of columns order_id, customer_id, region on respective tables, and that order_date is DATE or TIMESTAMP.
- Semantics: treated all joins as INNER because the original WHERE (plus business context) implies only orders with both customers and line items should be returned.
- Query intent: assumed the dashboard needs core order, item, and regional customer details rather than all columns; hence selective columns chosen.
- Performance assumptions: assumed join keys and region/order_date have cardinality/partitioning or clustering that can accelerate filtered scans; avoid SELECT to reduce I/O and network transfer.
- Safety: avoided destructive operations and stayed within Snowflake SQL.

Engineering counterpart
3:18 AM - Responded in 0s
What would you verify before shipping this to production?

Agent response
3:18 AM - Responded in 2s
Before shipping, I'd verify:
- Business logic: confirm join keys and filters match requirements (customer_id and order_id mappings, date inclusivity, EMEA definition).
- Data model tests: run EXPLAIN to confirm join order/types and that partition/clustering on order_date and region are used; check join cardinalities with small samples.
- Result validation: compare row counts and key metrics against a known baseline or the prior query window.
- Performance: execute on representative data volumes; monitor warehouse credits, runtime, and queueing in ANALYTICS_WH.
- Object existence: ensure referenced columns and tables exist and the service account has only necessary read privileges.
- Edge cases: test for NULL join keys, out-of-range dates, customers without region, and duplicate keys; ensure results are deterministic.
- Explain plan hygiene: verify no cross joins, no full table scans on large tables, and that predicates are pushed to scans.
- Observability: add query comments with ticket/owner and set query tags; optionally use result caching or a materialized view if patterns repeat.
- Governance & rollout: peer review, UAT sign-off, backout plan, and staged rollout with monitoring/alerts.

Engineering counterpart
3:18 AM - Responded in 0s
What are the top risks, and what's the rollback plan if this doesn't work?

Agent response
3:18 AM - Responded in 2s
Top risks
- Wrong join semantics: if join keys differ from assumptions (e.g., missing or composite keys), rows can be under- or over-counted.
- Performance not improved: predicate pushdown or clustering may be insufficient on ANALYTICS_WH, yielding similar or worse runtime/credits.
- Data quality issues: NULL join keys or sparse region values produce missing rows or duplicates.
- Object/column mismatch: renamed or missing tables/columns cause runtime errors.
- Permission gaps: the service account lacks usage on warehouse or read access to objects.
- Concurrency/warehouse contention: heavier joins peak load and queue in shared warehouses.

Rollback plan
- Abort and revert: cancel the job (via UI/CLI), and disable/suspend the dashboard or revert to the prior query version.
- Scripted rollback: keep the pre-change SQL (original SELECT) stored in version control; redeploy that text immediately.
- Infrastructure rollback: if a new warehouse or network rule was introduced, revert to the previous warehouse size/type or network policy.
- Monitoring: re-enable prior alert thresholds and watch runtime/credits; if the baseline degrades, fail back to the original query and notify stakeholders.

Engineering counterpart
3:18 AM - Responded in 0s
Thanks - reasoning on joins and selectivity is clear. Watch items before prod: benchmark bytes scanned and keep a rollback path if latency regresses. Review complete - no further response needed.