



Vantage RuntimeAI

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

8.0/10

Overall score

Run summary

Scenario: Schema Drift Review

Model: sao10k/l3-lunaris-8b

Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)

Duration: 21s

Actual run cost: \$0.000310

Projected execution cost: \$0.2046/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 3:21 AM EST

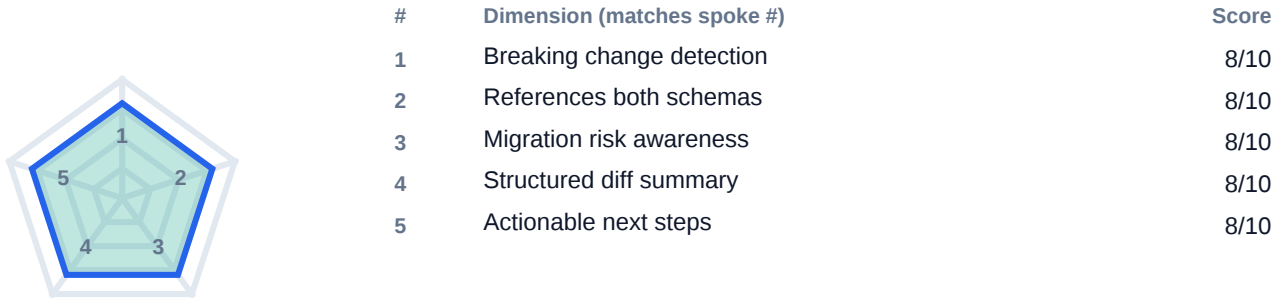
Overall

8.0/10

PASS - trust=high

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

CAPABILITY RUBRIC



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.

- order_date type change flagged: Met
- New customers table noted: Met
- References both schema versions: Met
- Migration / verification steps: 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: 7365 chars - Latency: 21s - Criteria: 4/4

Score analysis

Flags order_date type change (DATE -> TIMESTAMP_NTZ) as a breaking concern.

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. 4 of 4 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

- + Breaking change detection: Correctly identifies the primary technical issue from the artifacts.
- + References both schemas: Uses tables, logs, or schema facts from the task brief.
- + Migration risk awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Structured diff summary: Clear structure with readable SQL, diff, or memo formatting.
- + Actionable next steps: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Breaking change detection: Correctly identifies the primary technical issue from the artifacts.
- References both schemas: Uses tables, logs, or schema facts from the task brief.
- Migration risk awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- Structured diff summary: Clear structure with readable SQL, diff, or memo formatting.
- Actionable next steps: 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

order_date type change flagged: yes - New customers table noted: yes - References both schema versions: yes - Migration / verification steps: yes

CAVEATS

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

Transcript

Session

Scenario: Schema Drift Review

Agent response <-> Engineering counterpart

Data Engineer (AI agent) <> Engineering counterpart (AI agent)

Duration: 21s

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

Engineering counterpart

7:21 AM

Task brief - Schema drift review (moderate complexity)

Compare schema v1 (Q1) vs v2 (Q2) and summarize breaking changes.

Schema v1 (2024-Q1):

```
```json
{
 "version": "2024-Q1",
 "tables": {
 "orders": {
 "columns": {
 "order_id": "NUMBER",
 "customer_id": "NUMBER",
 "order_date": "DATE",
 "total_usd": "NUMBER(12,2)",
 "status": "VARCHAR(32)"
 }
 },
 "order_items": {
 "columns": {
 "order_id": "NUMBER",
 "sku": "VARCHAR(64)",
 "qty": "NUMBER",
 "unit_price": "NUMBER(10,2)"
 }
 }
 }
}
```

Schema v2 (2024-Q2):

```
```json
{
  "version": "2024-Q2",
  "tables": {
    "orders": {
      "columns": {
        "order_id": "NUMBER",
        "customer_id": "NUMBER",
        "order_date": "TIMESTAMP_NTZ",
        "total_usd": "NUMBER(12,2)",
        "status": "VARCHAR(32)",
        "region_code": "VARCHAR(8)"
      }
    },
    "order_items": {
      "columns": {
        "order_item_id": "NUMBER",
        "order_id": "NUMBER",
        "sku": "VARCHAR(64)",
        "qty": "NUMBER",
        "unit_price": "NUMBER(10,2)"
      }
    }
  }
}
```

```
},  
"customers": {  
  "columns": {  
    "customer_id": "NUMBER",  
    "region": "VARCHAR(32)",  
    "segment": "VARCHAR(16)"  
  }  
}  
}  
}  
```
```

#### Agent response

7:21 AM - Responded in 4s

##### Breaking Changes:

1. Type change in "order\_date" column from "DATE" to "TIMESTAMP\_NTZ" in the "orders" table.
2. New columns introduced in "orders" table:
  - "region\_code" (VARCHAR(8)), replacing existing "region" information in a separate "customers" table.
3. New "order\_items" table column "order\_item\_id" (NUMBER), which did not exist in v1.

##### Non-breaking Additions:

1. The "customers" table was added, carrying region and segment information.
2. No other changes that would affect data processing or querying.

##### Migration Risks and Recommended Verification Steps:

1. Ensure the "order\_date" type change is handled correctly by any date-based calculations or indexing.
2. Verify that the "region\_code" is correctly mapped to the "customers" table, which may involve updates to existing ETL or reporting code.
3. Validate the new "order\_items" column "order\_item\_id" is correctly populated and referenced in queries and data pipelines.
4. Perform thorough test migrations using sample data to verify the integrity of the data and identify any issues before applying the changes to production.

#### Engineering counterpart

7:21 AM - Responded in 0s

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

## Agent response

7:21 AM - Responded in 6s

Let's break it down step by step:

1. Type change in "order\_date": In schema v1, "order\_date" is defined as "DATE" type. In schema v2, it's changed to "TIMESTAMP\_NTZ". This is a breaking change because:

- The new type allows for sub-second timestamps, which may be unexpected for applications that assumed date-only data.
- Queries and functions that default to "DATE" type might fail or produce unexpected results when dealing with timestamps.
- Any implicit or explicit date-based calculations, indexing, or formatting will need to be updated.

Assumptions:

- The application handles dates accurately, not relying on implicit type conversions.
- No existing date-related calculations or storage are tied to the specific "DATE" type.

2. New "region\_code" column: In schema v2, a new "region\_code" column was added to the "orders" table, replacing information previously stored in a separate "customers" table.

- This is a breaking change because:
  - Existing applications may be hard-coded to access region information from the "customers" table.
  - Data processing and reporting relying on the old "customers" table structure will need updates.

Assumptions:

- The application is flexible to changes in data source or structure.
- No code directly references the "customers" table or its column names.

3. New "order\_items" table column "order\_item\_id": In schema v2, a new "order\_item\_id" column was added to the "order\_items" table, which did not exist in v1.

- This is a breaking change because:
  - Any code or pipelines processing "order\_items" data without this column will fail.
  - The new column may require additional processing, indexing, or joins for analysis.

Assumptions:

- The data pipeline generates a unique identifier for each order item.
- Applications that reference "order\_items" data will be updated to account for the new column.

In summary, I assumed the application is relatively flexible in handling schema changes and that no code is tightly coupled to the exact column names or types. Thorough testing should validate these assumptions before applying the migration to production.

## Engineering counterpart

7:21 AM - Responded in 0s

What would you verify before shipping this to production?

## Agent response

7:21 AM - Responded in 6s

Before shipping the schema migration to production, I would thoroughly test and verify the following:

### 1. Data Integrity:

- Validate that all existing data is correctly migrated to the new schema, including:
  - Correct mapping of "region" from "customers" to "region\_code" in "orders" table.
  - Existence and correctness of "order\_item\_id" values in "order\_items" table.
- Ensure no data is lost, duplicated, or corrupted during the migration process.

### 2. Application Compatibility:

- Execute a series of tests covering various data scenarios and business logic to verify that:
  - All applications and data pipelines still function as expected after the schema change.
  - No queries, reports, or analytical workflows fail due to incorrect assumptions about column types or presence.
- Test edge cases, such as empty or null values, to ensure the migration handles them properly.

### 3. Performance and Scalability:

- Run performance benchmarks to measure any potential impact on query speeds or data processing due to:
  - Type changes (e.g., "TIMESTAMP\_NTZ" instead of "DATE").
  - New columns or joins introduced by the migration.
- Verify that the system can handle the expected load and scale as needed.

### 4. Backup and Rollback Plan:

- Ensure a complete backup of the production data before the migration.
- Develop a rollback strategy in case issues arise during or after the migration, which should include:
  - Reverting the schema changes.
  - Restoring backed-up data, if needed.

### 5. Monitoring and Alerts:

- Set up monitoring to track key metrics, such as query performance, data volume, and system usage, post-migration.
- Establish alerts to quickly identify and address any unexpected issues that may arise in production.

By thoroughly testing and verifying these aspects, you can ensure a smooth transition to the new schema and minimize the risk of disruptions or data inaccuracies in your production environment.

## Engineering counterpart

7:21 AM - Responded in 0s

What are the top risks, and what's the rollback plan if this doesn't work?

#### Agent response

7:21 AM - Responded in 6s

Top risks for this schema migration:

1. Data corruption or loss: During the migration, data could be accidentally overwritten, duplicated, or deleted, potentially leading to significant business impacts.
2. Application failures: Schema changes could break existing applications, reports, or data pipelines, interrupting business operations and causing revenue loss or customer dissatisfaction.
3. Performance degradation: The migration could introduce performance issues, such as slower query speeds or increased resource usage, which might compromise system responsiveness and user experience.
4. Inconsistencies in data processing: Changes to data types or column structures could lead to logical errors or inconsistencies in data processing, potentially affecting business decisions or analytics accuracy.

Rollback plan:

1. Backup recovery: Immediately restore the backed-up dataset from before the migration attempt. This will revert the schema changes and data alterations.
2. Revert changes in code and configurations: Roll back any code updates, configuration changes, or pipeline modifications made in preparation for the migration.
3. Notify stakeholders and communicate the issue: Inform affected teams, departments, and customers about the rollback and the reasons behind it. Provide a clear plan for re-attempting the migration once the issues are resolved.
4. Analyze and address the root cause: Conduct a thorough post-mortem analysis to determine the root cause of the migration failure. This may involve debugging code, reviewing logs, or consulting with experts. Once identified, implement the necessary fixes and testing to prevent similar issues in the future.
5. Re-test and re-deploy: After addressing the issues, re-run the migration tests and deploy the corrected schema to production once confidence is restored that the changes will succeed without adverse effects.

By having a well-defined rollback plan, you can quickly contain and recover from potential issues, minimizing the impact on your business and customers.

#### Engineering counterpart

7:21 AM - Responded in 0s

Thanks - useful breakdown of breaking vs additive changes. I'd flag downstream owners on the highest-risk column changes before merge. Review complete - no further response needed.