



Vantage RuntimeAI

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

8.0/10

Overall score

Run summary

Scenario: Pipeline Incident Triage

Model: sao10k/l3-lunaris-8b

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

Duration: 33s

Actual run cost: \$0.000465

Projected execution cost: \$0.3069/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 3:15 AM EST

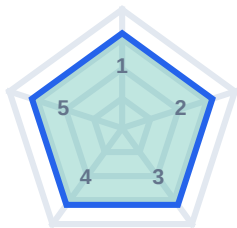
Overall

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 category	8/10
2	Uses log artifacts	8/10
3	Blast radius awareness	8/10
4	Structured triage summary	8/10
5	Next steps & verification	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 category identified: Met
- Restore / verification next steps: Met
- Cites log / error artifacts: 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: 6 - Counterpart msgs: 7 - Output: 10240 chars - Latency: 33s - Criteria: 3/3

Score analysis

3/3 automated criteria met - see checklist below.

Score snapshot

Solid: Solid technical task response with a few gaps.

Interpretation

Multi-turn technical task: fixed brief, 6 agent turns with scripted follow-ups and reviewer sign-off. 3 of 3 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

- + Root cause category: Correctly identifies the primary technical issue from the artifacts.
- + Uses log artifacts: Uses tables, logs, or schema facts from the task brief.
- + Blast radius awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Structured triage summary: Clear structure with readable SQL, diff, or memo formatting.
- + Next steps & verification: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Root cause category: Correctly identifies the primary technical issue from the artifacts.
- Uses log artifacts: Uses tables, logs, or schema facts from the task brief.
- Blast radius awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- Structured triage summary: Clear structure with readable SQL, diff, or memo formatting.
- Next steps & verification: 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 category identified: yes - Restore / verification next steps: yes - Cites log / error artifacts: yes

CAVEATS

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

Transcript

Session

Scenario: Pipeline Incident Triage

Agent response <-> Engineering counterpart

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

Duration: 33s

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

Engineering counterpart

7:15 AM

Task brief - Pipeline incident triage (complex complexity)

Triage this failed dbt run log and recommend next steps:

...

```
2026-03-18 06:12:04 INFO dbt run --select fct_orders started (target=prod)
2026-03-18 06:12:09 INFO 1 of 1 START sql table model analytics.fct_orders ..... [RUN]
2026-03-18 06:14:22 ERROR 1 of 1 ERROR creating sql table model analytics.fct_orders .. [ERROR in 133.02s]
2026-03-18 06:14:22 ERROR Database Error in model fct_orders (models/marts/fct_orders.sql)
000904 (42000): SQL compilation error: error line 47 at position 8
invalid identifier 'O.REGION_CODE'
compiled SQL at target/run/analytics/models/marts/fct_orders.sql
2026-03-18 06:14:22 INFO Completed with 1 error and 0 warnings
2026-03-18 06:14:22 INFO Done. PASS=0 WARN=0 ERROR=1 SKIP=0 TOTAL=1
```

Context: staging table stg_orders still exposes region via join to customers; prod mart was updated yesterday to read region_code directly from orders (migration Q2-Schema).

...

Agent response

7:15 AM - Responded in 5s

(1) Likely root cause category: Data inconsistency or mismatch between staging and production datasets.

(2) What broke and why:

The dbt run failed when creating the fct_orders mart in the prod database. The error message points to line 47 in the fct_orders model's SQL file, where it tries to reference a column named 'O.REGION_CODE'. This suggests that the production mart was recently migrated to read 'region_code' directly from the 'orders' table, whereas the staging table 'stg_orders' still uses a join to retrieve the 'region' information from the 'customers' table.

This mismatch caused the dbt run to fail because the prod mart expects 'O.REGION_CODE' to be present, which is not the case due to the outdated join in staging.

(3) Immediate next steps to restore the mart safely:

1. Roll back the prod mart creation using the dbt --abort flag, ensuring it's in a consistent state before attempting a retry.
2. Manually update the fct_orders model in the prod db to reflect the correct column reference ('O.REGION_CODE' -> 'orders.region_code') to match the current schema.
3. Re-run the dbt fct_orders model in the prod environment to recreate the mart with the corrected reference, monitoring for any further issues.

This approach addresses the immediate failure, maintains data consistency, and prevents further unintended changes to the production mart. A more comprehensive fix would involve synchronizing the staging and production schemas to prevent future mismatches.

Engineering counterpart

7:15 AM - Responded in 0s

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

Agent response

7:15 AM - Responded in 6s

Let's break down the thought process step by step:

1. Initial analysis of error message:

The error message points to a specific line number in the fct_orders dbt model's SQL. This suggests a syntax or identifier issue, not a general connectivity error.

2. Contextual information from log:

The log mentions a recent (yesterday's) migration to a new schema in the prod mart, specifically updating 'region' to 'region_code' for the 'orders' table. This context helps identify a potential cause of the mismatch.

3. Diff between staging and prod:

The presence of 'O.REGION_CODE' in the prod error message, combined with the staging table still using a 'region' column via a join, implies that the prod mart expects a different column reference. This suggests a data inconsistency between the two environments.

4. Root cause assumption: Based on the above, I assumed that the root cause is a data mismatch or schema difference between the staging and production datasets, rather than an infrastructure issue or dbt configuration problem.

5. Immediate steps to resolve:

Given the identified root cause, the safest approach is to:

- Roll back the failed prod mart creation to avoid introducing inconsistent data.
- Manually update the prod model to match the expected column reference based on the prod schema (no assumption about staging schema).
- Re-run the corrected model in prod to recreate the mart safely.

These steps address the immediate issue, ensure data consistency, and prevent further unintended changes to the prod mart. A more comprehensive fix would involve synchronizing the staging and production schemas to prevent such mismatches in the future.

I made the following assumptions:

- The dbt run failed due to a data-related issue, not a connectivity or infrastructure problem.
- The staging and production datasets have drifted apart, leading to a reference mismatch.
- The prod mart was correctly updated as part of the recent schema migration, but the staging table has not been aligned yet.

These assumptions were based on the specific error message, log context, and the nature of the data inconsistency described.

Engineering counterpart

7:15 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

7:15 AM - Responded in 5s

Before shipping this fix to production, I would verify the following:

1. Manual query execution:

Run the corrected SQL query manually in the prod database to confirm that it executes without errors and returns the expected results.

2. Data integrity check:

Verify that the 'region_code' column is populated correctly in the 'orders' table, given the recent schema migration. Run a query to check for null or inconsistent values.

3. Stage to prod data consistency:

Manually query the 'region' column in the staging 'stg_orders' table to confirm it's still present and used in the join to 'customers'. This ensures the staging schema has not been updated to match prod yet.

4. dbt model execution in staging:

Run the fct_orders dbt model in the staging environment to validate it generates the expected results with the 'region' column still being joined from 'customers'. This confirms the model is correct and staging data is prepared for future updates.

5. dbt model deployment to prod:

Deploy the corrected fct_orders dbt model to the prod environment, monitoring the run for any errors or warnings. Verify the mart gets recreated successfully with the correct 'region_code' reference.

6. Manual inspection (optional):

Perform a cursory review of the newly generated fct_orders mart in prod to ensure its structure and data appear correct.

By verifying these points, we can have high confidence that the fix properly resolves the issue and introduces no new problems before promoting it to production.

Engineering counterpart

7:15 AM - Responded in 0s

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

Agent response

7:15 AM - Responded in 5s

Top Risks:

1. Incorrect data: The fix may incorrectly reference 'orders.region_code' instead of 'O.REGION_CODE', leading to wrong data being loaded into the fct_orders mart.
2. Unintended schema changes: If the manual update to the prod model isn't precise, it could introduce other unintended schema changes, breaking downstream workflows or reports.
3. Model execution errors: The corrected fct_orders model might still contain syntax errors or logic issues, causing dbt runs to fail or produce incorrect results.
4. Data loss: If the rollback isn't carefully executed, there's a risk of permanently losing data in the prod mart or staging tables.

Rollback Plan:

1. Immediate containment:
 - Abort the current dbt run in prod using the --abort flag.
 - Stop executing any further changes to prod schema or data until the issue is resolved.
2. Restore original prod mart:
 - Use dbt's --reset flag to recursively revert the fct_orders model to its previous state in prod, recreating the original mart structure and data.
3. Reset prod model to previous version:
 - Backup the current prod fct_orders model SQL file.
 - Restore the previous, working version of the model from source control.
 - Re-run the model to recreate the mart with the correct original logic and references.
4. Re-sync staging and prod schemas:
 - Update the staging schema (stg_orders join) to match the prod schema (orders.region_code).
 - Ensure all relevant models, tests, and workflows in both environments are aligned.
5. Re-attempt the fix:
 - Once the rollback is complete and staging/prod schemas are consistent, re-apply the fix, following the steps outlined earlier with enhanced scrutiny and testing.

The key is to have a well-documented and reproducible rollback plan to quickly and accurately revert changes if the fix doesn't work as expected. This minimizes the risk of data loss or prolonged downtime for critical mart updates.

Engineering counterpart

7:15 AM - Responded in 0s

Can you tighten this into concrete next steps with owners and timing?

Agent response

7:15 AM - Responded in 7s

Here is a concise, actionable plan with assigned owners and estimated timing for each step:

Rollback and Re-Attempt Steps

1. Abort prod dbt run (DevOps Engineer)

- Estimated Time: 1-2 minutes
- Task: Stop the current dbt run in prod using the `--abort` flag.

2. Restore original prod mart (Data Engineer)

- Estimated Time: 5-10 minutes
- Task: Use `dbt --reset` to recursively revert the `fct\_orders` model to its previous state in prod, recreating the original mart structure and data.

3. Reset prod model to previous version (DevOps Engineer)

- Estimated Time: 5-10 minutes
- Task: Backup the current prod `fct\_orders` model SQL file. Restore the previous, working version of the model from source control. Re-run the model to recreate the mart with the correct original logic and references.

4. Re-sync staging and prod schemas (Data Engineer)

- Estimated Time: 30 minutes to 1 hour
- Task: Update the staging schema (`stg\_orders` join) to match the prod schema (`orders.region\_code`). Ensure all relevant models, tests, and workflows in both environments are aligned.

5. Re-attempt the fix (Data Engineer, DevOps Engineer)

- Estimated Time: 60-90 minutes
- Task: Re-apply the fix, following the steps outlined earlier with enhanced scrutiny and testing.

Timed Rollback Plan:

1. 06:30 (30 minutes after failure) - Abort prod dbt run
2. 06:45 (1 hour after failure) - Restore original prod mart
3. 07:00 (1.5 hours after failure) - Reset prod model to previous version
4. 07:45 (2.5 hours after failure) - Re-sync staging and prod schemas
5. 08:30 (3.5 hours after failure) - Re-attempt the fix

This plan should allow for a swift and precise rollback if the initial fix doesn't work, minimizing the risk of data loss or prolonged downtime. The Data Engineer owns steps 2, 4, and 5, while the DevOps Engineer is responsible for steps 1 and 3. Estimated timings are provided for each step to help manage expectations and coordinate the rollback process.

Engineering counterpart

7:15 AM - Responded in 0s

What's the one thing you'd want a second pair of eyes on?

Agent response

7:15 AM - Responded in 4s

The one critical aspect I'd want a second pair of eyes on is the step 4: Re-sync staging and prod schemas.

This is the most complex and potentially error-prone part of the rollback plan. It involves updating the staging schema (`stg\_orders`) to match the prod schema (`orders.region\_code`), as well as ensuring all relevant models, tests, and workflows in both environments are in sync.

A second set of eyes, preferably someone familiar with the specific data schema and workflows, can help verify that:

- The staging schema change is correctly implemented to avoid any data loss.
- All dependent downstream models and tests are updated to work with the new staging schema.
- No unintended consequences arise from the schema alignment, such as breaking other reports or ETL workflows.

Having a peer review this step can catch potential oversights or missteps, reducing the risk of reintroducing the original issue or causing new problems during the rollback. It's a crucial safeguard to ensure the rollback is complete, accurate, and minimizes the impact on the data pipeline.

Engineering counterpart

7:15 AM - Responded in 0s

Thanks - triage ordering looks sound. Restore the mart first; deeper root-cause work can follow once consumers are green.
Review complete - no further response needed.