



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: google/gemma-4-31b-it

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

Duration: 3m 1s

Actual run cost: \$0.001614

Projected execution cost: \$1.07/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 1:13 AM EST

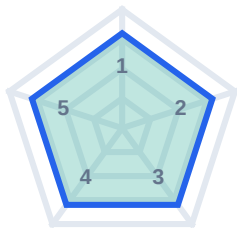
Overall

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: 13388 chars - Latency: 181s - 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: 3m 1s

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

Engineering counterpart

5:10 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

5:10 AM - Responded in 10s

1. Likely Root Cause Category

Schema Mismatch / Breaking Change (Upstream Dependency)

2. What broke and why

The model `fct\_orders` failed during compilation/execution because it attempted to reference a column (`O.REGION\_CODE`) that does not exist in the source table/view currently provided by the `orders` relation.

Reference: `000904 (42000): SQL compilation error: error line 47 at position 8 invalid identifier 'O.REGION\_CODE'`

Reasoning: The "migration Q2-Schema" updated the mart to pull `region\_code` directly from the orders table instead of joining via customers. However, the underlying physical table in the production environment has not yet been updated to include this column, leading to an "invalid identifier" error when the SQL is executed.

3. Immediate Next Steps to Restore Mart

To restore the mart safely without waiting for a full database migration:

1. Revert Code: Roll back the changes in `models/marts/fct\_orders.sql` to the previous version that retrieved the region via the join to the customers table (as noted in the context that `stg\_orders` still exposes this).
2. Deploy & Run: Deploy the reverted code and trigger a `dbt run --select fct\_orders` to restore the production table to its last known working state.
3. Sync Migration: Coordinate with the DBA/Data Platform team to verify when the `REGION\_CODE` column will be physically added to the production `orders` table before re-attempting the Q2-Schema migration.

Engineering counterpart

5:10 AM - Responded in 0s

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

Agent response

5:10 AM - Responded in 21s

To arrive at the triage result, I followed a deductive process moving from the hard evidence (logs) to the environmental context (brief) to the technical logic (SQL behavior).

Step-by-Step Reasoning

Step 1: Analyze the Error Type

I first looked at the error code `000904 (42000)`. In Snowflake (and similar warehouses), a `42000` code is a syntax or compilation error. Specifically, `invalid identifier 'O.REGION\_CODE'` tells me that the SQL is syntactically correct, but the database engine cannot find a column named `REGION\_CODE` in the table aliased as `O`.

Step 2: Identify the Point of Failure

The log specifies `error line 47 at position 8`. This confirms the failure isn't a connection issue or a permission error (which would happen at the start of the run); it is a failure during the execution of the specific SELECT statement used to build `fct\_orders`.

Step 3: Connect to the Provided Context

I compared the error to the provided context:

The Change: The mart was updated to read `region\_code` directly from `orders`.

The Current State: The error is specifically about `REGION\_CODE`.

The Conflict: The context mentions that `stg\_orders` (the staging layer) still uses a join to customers. This suggests a "version mismatch" where the dbt code (the "request") is ahead of the physical database schema (the "reality").

Step 4: Determine the "Why"

If the code was updated yesterday but is failing today, and the error is a missing column, the logical conclusion is that the dbt deployment happened, but the underlying DDL (Data Definition Language) change to add the `REGION\_CODE` column to the production `orders` table either failed, was skipped, or is scheduled for a later window.

Step 5: Formulate the Recovery Strategy

Since this is a production failure, the priority is MTTR (Mean Time To Recovery).

I know the "old way" (joining to customers) still works because the context states `stg\_orders` still exposes that data.

Therefore, the fastest path to a green pipeline is a code revert, not a database migration (which usually requires more approvals and carries higher risk).

Assumptions Made

To reach this conclusion, I made the following technical assumptions:

1. Alias Mapping: I assumed that `O` in `O.REGION\_CODE` refers to the `orders` table/CTE, given the context of the "Q2-Schema" migration.
2. Deployment Sequence: I assumed that the dbt code was deployed to production before the physical table schema was updated. In a perfect CI/CD pipeline, these would be atomic or sequenced (Schema change $\rightarrow$ Code change), but the logs suggest they were decoupled.
3. Upstream Availability: I assumed that the `customers` table still contains the region data, as the context explicitly mentioned `stg\_orders` still exposes it.
4. Environment Consistency: I assumed the `target=prod` environment is where the mismatch exists, as the failure occurred during the `prod` run.

Engineering counterpart

5:10 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

5:11 AM - Responded in 20s

Before shipping the fix (either the revert or the schema update) to production, I would perform these four verification steps to ensure I am not trading one failure for another.

1. Validate the "Old Path" still exists

Since the proposed fix is to revert to the join-based logic, I must verify that the join path is still viable.

The Check: Run a manual query in the production environment:

```
`SELECT region FROM analytics.stg_orders LIMIT 10;` (or whatever the join logic was).
```

Why: If the `customers` table was also modified as part of the Q2-Schema migration, reverting the code might lead to a different "invalid identifier" error. I need to confirm the "safety net" still exists.

2. Verify Data Parity (The "Silent Failure" Check)

A compilation error is loud, but a logic error is silent. If I revert to the old join, I need to ensure the data is still accurate.

The Check: Compare a small sample of `region` values from the old join method against the intended `region\_code` values from the new schema (if the column exists in a dev/stage environment).

Why: If the definition of "Region" changed during the Q2 migration (e.g., moving from `North America` to `NA`), reverting the code will restore the pipeline but introduce "stale" or "incorrect" data into the mart.

3. Check for Downstream Dependencies

`fct\_orders` is likely a base for other models (e.g., `dm\_regional\_sales`).

The Check: Run `dbt list --select fct\_orders+` to see everything that depends on this model.

Why: If any downstream models were also updated yesterday to expect `region\_code` instead of `region`, reverting `fct\_orders` will cause a "domino effect" of failures further down the DAG. I would need to revert the entire dependency chain, not just one model.

4. Environment Sync Confirmation

If the decision is to proceed with the new code rather than a revert, I must verify the DDL.

The Check: Describe the table in production: `DESC TABLE orders;`

Why: I need to confirm exactly why the column is missing. Is it a naming discrepancy (e.g., `REG\_CODE` vs `REGION\_CODE`) or is the column entirely absent? This prevents a "trial-and-error" deployment cycle.

Summary Checklist for the PR/Change Request:

- [] Confirmed: Join path to `customers` is still active in Prod.
- [] Confirmed: No downstream models are strictly dependent on the new `region\_code` naming.
- [] Confirmed: Data values in the old path match the business requirements for today.
- [] Verified: The fix was tested in a `dev` or `staging` environment that mirrors Prod schema.

Engineering counterpart

5:11 AM - Responded in 0s

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

Agent response

5:11 AM - Responded in 32s

Since we are operating in a production environment during an outage, we must weigh the risk of incorrect data (silent failure) against the risk of no data (hard failure).

Top Risks

1. The "Domino Effect" (Downstream Breaking Changes)

The biggest risk is that ``fct_orders`` is not the only model that was updated. If downstream models (e.g., ``fct_sales_by_region``) were also updated to expect the new ``region_code`` column, reverting only ``fct_orders`` will fix the first error but trigger a series of new "invalid identifier" errors immediately after.

Impact: Prolonged outage; "whack-a-mole" debugging in production.

2. Data Semantic Drift (Silent Corruption)

If the Q2-Schema migration changed how regions are defined (e.g., changing a mapping from ``Region Name`` $\rightarrow$ ``Region Code``), reverting to the old join will provide the old format.

Impact: The pipeline turns "green," but reports are wrong. Business users may make decisions based on mismatched regional data.

3. Circular Dependency/State Conflict

If the production environment has "partial" migrations (some tables updated, others not), a revert might conflict with other models that have already successfully migrated to the new schema.

Impact: Inconsistent state across the ``analytics`` schema.

Rollback Plan

If the fix (the revert) is deployed and the pipeline still fails or data is corrupted, we execute the following tiered rollback strategy:

Tier 1: Immediate Code Rollback (The "Undo" Button)

If the revert causes new errors (Risk #1), we immediately roll back the Git commit to the state as of two days ago (before the Q2-Schema migration started).

Action: ``git revert <commit_id_of_migration>``

Goal: Restore the entire project to the last known "Global Stable State" rather than patching individual models.

Tier 2: Point-in-Time Recovery (The "Nuclear" Option)

If the dbt run succeeded but we discover data corruption (Risk #2) and the ``fct_orders`` table was overwritten (as is common with ``table`` materialization), we cannot simply "revert code" to fix the data.

Action: Use Snowflake/Database Time Travel to restore the table to its state before yesterday's failed run.

Example: ``CREATE OR REPLACE TABLE analytics.fct_orders CLONE analytics.fct_orders AT (OFFSET => -86400);`` (restores to 24 hours ago).

Goal: Restore data integrity immediately while the engineering team fixes the code in a sandbox.

Tier 3: Emergency Hotfix (The "Bypass")

If the join is broken and the new column is missing, and we cannot restore from backup, we implement a "hardcoded constant" or "null" column.

Action: Replace ``O.REGION_CODE`` with ``CAST(NULL AS STRING) AS region_code`` in the SQL.

Goal: Allow the pipeline to complete so that other critical (unrelated) parts of the business mart are updated, accepting that regional reporting will be empty until the schema is fixed.

Engineering counterpart
5:11 AM - Responded in 0s

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

Agent response

5:13 AM - Responded in 1m 32s

Here is the tightened Incident Action Plan. This assumes a P1/High Priority status given the production failure.

Incident Action Plan: `fct\_orders` Failure

Goal: Restore `fct\_orders` availability with minimal data drift.

Strategy: Revert to the stable "Join-based" logic while verifying downstream impact.

| Timing | Action Item | Owner | Success Criteria |

| :--- | :--- | :--- | :--- |

| T + 0min | Impact Analysis: Run `dbt list --select fct\_orders+` to identify all downstream dependencies. | Data Engineer | List of all affected downstream models identified. |

| T + 15min | Schema Validation: Run `DESC TABLE orders` and `SELECT region FROM stg\_orders LIMIT 1` in Prod. | Data Engineer | Confirmation that the "old" join path is still physically available. |

| T + 30min | Code Revert: Revert `fct\_orders.sql` (and any identified downstream models) to the pre-Q2-Schema commit. | Analytics Engineer | PR submitted and peer-reviewed. |

| T + 45min | Production Deploy: Merge and trigger `dbt run --select fct\_orders+`. | DevOps / DE | Pipeline completes with `PASS=X, ERROR=0`. |

| T + 60min | Data QA: Compare row counts and a sample of `region` values against yesterday's snapshot. | QA / Analyst | Data parity confirmed; no unexpected NULLs. |

Escalation & Rollback Triggers

If the following occurs, pivot immediately:

Trigger A: The revert causes new "invalid identifier" errors in downstream models.

Action: Abandon model-level revert $\rightarrow$ Full Project Rollback to last stable Git tag.

Trigger B: The revert succeeds, but data values are fundamentally different (Semantic Drift).

Action: Trigger Time Travel Restore to recover the table state from 24 hours ago.

Trigger C: The "old" join path in `stg\_orders` is also missing.

Action: Implement Emergency Hotfix (Null column) to unblock the pipeline for other models.

Post-Incident Requirement

Owner: Data Architect

Deliverable: Root Cause Analysis (RCA) specifically addressing why the DDL (schema change) and DML (dbt code) were not deployed as an atomic transaction.

Engineering counterpart
5:13 AM - Responded in 0s

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

Agent response

5:13 AM - Responded in 7s

The one thing I would absolutely want a second pair of eyes on is the Downstream Dependency Mapping (The "Domino Effect").

Here is why:

The error in ``fct_orders`` is a hard failure (the run stopped), which is easy to see. However, if other models were updated to expect ``region_code`` and they are scheduled to run after ``fct_orders`` or are part of a larger ``dbt build``, they might not have failed yet-or they might fail in a way that is harder to debug.

If I revert ``fct_orders`` to provide ``region`` (the old name), but a downstream model ``dm_regional_sales`` is now looking for ``region_code`` (the new name), I haven't fixed the pipeline-I've just shifted the failure point one step down the DAG.

I would ask the second engineer specifically:

> "I'm reverting ``fct_orders`` to use the old ``region`` column. Can you double-check the Git history for all models that depend on ``fct_orders`` to ensure none of them were updated to expect ``region_code``? If they were, we need to revert them in the same PR, or we're just moving the error."

Engineering counterpart

5:13 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.