



Vantage RuntimeAI

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

8.0/10

Overall score

Run summary

Scenario: Freshness SLA Breach

Model: google/gemma-4-31b-it

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

Duration: 1m 32s

Actual run cost: \$0.001076

Projected execution cost: \$1.18/mo (50/biz day × 22 days)

Report date: July 14, 2026 - 1:54 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	Stale upstream identified	8/10
2	Uses SLA / freshness facts	8/10
3	Downstream blast radius	8/10
4	Structured status summary	8/10
5	Restore & verify steps	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.

- Upstream blocker identified: Met
- SLA breach acknowledged: Met
- Downstream impact noted: Met
- Restore steps listed: 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: 10513 chars - Latency: 92s - Criteria: 4/4

Score analysis

Identifies int_order_enriched failure as upstream blocker.

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)

- + Stale upstream identified: Correctly identifies the primary technical issue from the artifacts.
- + Uses SLA / freshness facts: Uses tables, logs, or schema facts from the task brief.
- + Downstream blast radius: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Structured status summary: Clear structure with readable SQL, diff, or memo formatting.
- + Restore & verify steps: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Stale upstream identified: Correctly identifies the primary technical issue from the artifacts.
- Uses SLA / freshness facts: Uses tables, logs, or schema facts from the task brief.
- Downstream blast radius: Shows appropriate safety, budget, or risk awareness for the scenario.
- Structured status summary: Clear structure with readable SQL, diff, or memo formatting.
- Restore & verify 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

Upstream blocker identified: yes - SLA breach acknowledged: yes - Downstream impact noted: yes - Restore steps listed: yes

CAVEATS

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

Transcript

Session

Scenario: Freshness SLA Breach

Agent response <-> Engineering counterpart

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

Duration: 1m 32s

Engineering counterpart

5:52 AM

Task brief - Freshness SLA breach (moderate complexity)

A P1 mart breached its freshness SLA. Triage using the metadata JSON:

```
```json
{
 "mart": "analytics.fct_orders",
 "owner_team": "data-platform",
 "sla": {
 "max_staleness_minutes": 120,
 "tier": "P1 revenue dashboard"
 },
 "freshness": {
 "last_successful_run_utc": "2026-07-05T09:12:00Z",
 "now_utc": "2026-07-05T15:40:00Z",
 "staleness_minutes": 388,
 "status": "BREACHED"
 },
 "upstream": [
 {"model": "staging.stg_orders", "last_run_utc": "2026-07-05T15:05:00Z", "status": "success"},
 {"model": "staging.stg_customers", "last_run_utc": "2026-07-05T14:58:00Z", "status": "success"},
 {"model": "intermediate.int_order_enriched", "last_run_utc": "2026-07-05T09:12:00Z", "status": "failed", "error":
"DBT_RUN_TIMEOUT after 45m"}
],
 "downstream_consumers": ["Looker Revenue Overview", "exec_daily_orders Slack bot"],
 "note": "int_order_enriched has been failing since morning; fct_orders has not refreshed."
}
```
```

Agent response

5:53 AM - Responded in 19s

Incident Report: Freshness SLA Breach - `analytics.fct\_orders`

Blocking Upstream Model

The refresh is blocked by `intermediate.int\_order\_enriched`.

Status: Failed

Root Cause: `DBT\_RUN\_TIMEOUT` (Process exceeded 45m limit).

Impact: This model has been stalled since 09:12 UTC, preventing the final mart from updating.

Blast Radius

The breach currently impacts the following high-visibility downstream consumers:

1. Looker Revenue Overview: Stale data appearing on the P1 Revenue Dashboard.
2. exec_daily_orders Slack bot: Executive stakeholders are receiving outdated order metrics.

Restoration Plan

To restore the mart within the SLA, execute the following steps in order:

1. Increase Timeout/Resource Allocation: Temporarily increase the timeout threshold for `intermediate.int\_order\_enriched` or allocate more warehouse resources (e.g., scale up the compute cluster) to resolve the `DBT\_RUN\_TIMEOUT`.
2. Manual Trigger of Intermediate Model: Manually trigger a run for `intermediate.int\_order\_enriched` and monitor until completion.
3. Trigger Downstream Mart: Once the intermediate model succeeds, trigger a refresh of `analytics.fct\_orders`.
4. Verification: Confirm `staleness\_minutes` is < 120 and verify data consistency in the Looker Revenue Overview dashboard.
5. Post-Mortem: Investigate the cause of the timeout (e.g., data volume spike or inefficient join) to implement a permanent fix.

Engineering counterpart

5:53 AM - Responded in 0s

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

Agent response

5:53 AM - Responded in 21s

To arrive at the resolution, I followed a systematic triage process used in data reliability engineering. Here is the step-by-step reasoning and the assumptions I made.

Step-by-Step Reasoning

1. Quantifying the Breach

First, I looked at the `freshness` block. The `now\_utc` is 15:40 and the last success was 09:12. The staleness is 388 minutes, which is more than 3x the allowed 120-minute SLA. This confirms a critical failure (P1) requiring immediate intervention.

2. Identifying the "Bottleneck" (Upstream Analysis)

I analyzed the `upstream` array to find the point of failure. I looked for the model with the oldest `last\_run\_utc` that also had a `failed` status:

 `stg\_orders` and `stg\_customers` both ran successfully within the last hour. They are healthy.

 `int\_order\_enriched` last succeeded at 09:12 (exactly when the mart stopped updating) and is currently in a `failed` state.

Conclusion: This is the "blocking" node. Since `fct\_orders` depends on this intermediate model, it cannot refresh until this specific failure is resolved.

3. Determining the Root Cause

I examined the `error` field for the blocking model. The error `DBT\_RUN\_TIMEOUT after 45m` tells me the issue isn't a syntax error or a schema change (which would be a `Compilation Error` or `Database Error`), but a performance/resource issue. The query is simply taking too long to execute.

4. Mapping the Blast Radius

I looked at the `downstream\_consumers` list. Because this is a "P1 revenue dashboard," I categorized the impact as "high visibility." This informs the urgency and the need to communicate specifically with the stakeholders of the Looker dashboard and the Slack bot.

5. Designing the Recovery Path

Since the cause is a timeout, I knew that simply "retrying" the job without changes would likely result in another timeout. Therefore, the restoration steps had to follow a specific logic:

 Fix the environment first (Increase timeout/resources).

 Clear the blockage (Run the intermediate model).

 Complete the pipeline (Run the final mart).

 Validate (Check the dashboard).

Assumptions Made

During this triage, I made the following assumptions:

Linear Dependency: I assumed a linear DAG (Directed Acyclic Graph) where `stg` $\rightarrow$ `int` $\rightarrow$ `fct`. While `fct\_orders` has multiple upstreams, I assumed the failure of any one of them completely blocks the final mart.

Resource Availability: I assumed that the data platform team has the permissions to temporarily scale up warehouse resources or modify DBT timeout settings in production.

Data Integrity: I assumed the timeout was caused by volume or compute constraints rather than a "runaway query" (like a Cartesian join). If it were a Cartesian join, increasing the timeout wouldn't help; the query would eventually crash the warehouse. I opted for the "increase resources" path as the fastest way to restore service before performing a deeper code audit.

Idempotency: I assumed the models are idempotent (i.e., running them again will overwrite the stale data correctly without creating duplicates).

Engineering counterpart
5:53 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

5:53 AM - Responded in 28s

Before shipping the fix (the resource increase and the manual trigger) to production, I would perform these specific checks to ensure I am not trading a freshness breach for a data quality disaster.

1. Query Plan Analysis (The "Cartesian Join" Check)

Before simply increasing the timeout, I would inspect the query plan for ``int_order_enriched``.

The Risk: If a recent code change introduced a cross-join (Cartesian product), increasing the timeout won't fix the problem-it will just allow the query to run longer, consume massive amounts of compute credits, and potentially crash the warehouse for all other users.

Verification: I would check for "Exploding Joins" (where the output row count is significantly higher than the input row count). If the query is logically sound but just slow due to volume, the resource increase is safe.

2. Data Volume Audit (The "Data Spike" Check)

I would compare the row counts of the upstream staging tables (``stg_orders``, ``stg_customers``) against historical averages.

The Risk: A sudden 10x spike in source data (e.g., a bot attack or a duplicated ingestion load) could be causing the timeout.

Verification: If there is a massive spike in source data, I would investigate if the data is "garbage" before refreshing. Refreshing a mart with corrupted/duplicated source data is worse than having stale data.

3. Idempotency & State Verification

I would verify how the model handles "re-runs."

The Risk: If ``int_order_enriched`` uses an ``INSERT INTO`` instead of a ``MERGE`` or ``REPLACE`` (Overwrite), running it manually after a partial failure could result in duplicate records.

Verification: I would confirm the materialization strategy (e.g., ``table`` or ``incremental``). If it is incremental, I would check if I need to perform a ``--full-refresh`` to clear any partial data written during the failed 45-minute window.

4. Downstream Dependency Lock-Check

I would check if any other P1/P0 models depend on ``int_order_enriched`` besides ``fct_orders``.

The Risk: Solving the problem for the Revenue Dashboard might trigger a cascade of other delayed models that could overload the warehouse if they all start running simultaneously.

Verification: I would review the full DAG to see if I should stagger the manual triggers to avoid a "thundering herd" problem on the compute cluster.

5. Resource Cost Impact

Since I proposed increasing the warehouse size or timeout:

The Risk: Scaling up a warehouse (e.g., moving from Small to Large in Snowflake) can significantly increase costs per hour.

Verification: I would ensure the scale-up is temporary and set a reminder (or an auto-suspend limit) to scale back down once the backlog is cleared.

Engineering counterpart
5:53 AM - Responded in 0s

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

Agent response

5:54 AM - Responded in 24s

In a P1 production environment, the primary goal is to balance speed of recovery with system stability. Here are the top risks associated with the proposed restoration plan and the corresponding rollback strategies.

Top Risks

1. The "Infinite Loop" / Resource Exhaustion Risk

Scenario: The timeout wasn't caused by "not enough resources," but by a logical error (e.g., a circular reference or a massive Cartesian join).

Impact: By increasing the timeout and warehouse size, we allow a "runaway query" to consume massive amounts of compute credits and potentially lock tables, causing a wider system outage (affecting other marts).

2. The "Data Corruption" Risk

Scenario: The failure occurred during a partial write, or the source data contains duplicates that caused the timeout.

Impact: Forcing the model to complete could result in `fct\_orders` containing duplicated revenue figures, leading to incorrect financial reporting for executives.

3. The "Thundering Herd" Risk

Scenario: Once `int\_order\_enriched` finally succeeds, it triggers a cascade of downstream dependencies.

Impact: The sudden surge in concurrent queries may overwhelm the warehouse, leading to a secondary wave of timeouts across other unrelated P1 dashboards.

Rollback & Contingency Plan

If the restoration steps fail or cause further instability, I will execute the following rollback plan:

Level 1: Immediate Technical Rollback (The "Kill Switch")

If the manual run exceeds the new timeout or warehouse CPU hits 100% for an extended period:

1. Kill the Query: Immediately terminate the PID of the running `int\_order\_enriched` job.
2. Revert Resources: Scale the warehouse back to its original size and reset the timeout limit to prevent further cost overruns.
3. Isolate the Model: Disable the scheduled trigger for this specific pipeline to prevent it from automatically retrying and crashing the system again.

Level 2: Data State Recovery (The "Time Travel" Plan)

If the manual run succeeds but data validation shows corruption (e.g., duplicate orders):

1. Restore from Snapshot: Use database "Time Travel" (if using Snowflake/BigQuery) or restore the `fct\_orders` table from the last known healthy backup (2026-07-05T09:12:00Z).
2. Clear Intermediate State: Drop the `int\_order\_enriched` table entirely to ensure no partial/corrupted data remains.

Level 3: Stakeholder Communication (The "Expectation" Plan)

If technical recovery is not possible within the next 60 minutes:

1. Declare "Degraded State": Formally notify the owners of the Looker Revenue Overview and the Slack bot that the dashboard is "Degraded" rather than "Updating."
2. Provide ETA: Move from "fixing" to "investigating," providing a new ETA based on a code audit rather than a resource increase.
3. Manual Workaround: If urgent, provide executives with a raw query or a CSV export of the most critical metrics from the staging layer, bypassing the broken intermediate model.

Engineering counterpart

5:54 AM - Responded in 0s

Thanks - upstream blocker and blast radius are well ordered. Prioritize revenue-facing dashboards if full freshness slips past the hour mark. Review complete - no further response needed.