



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-3-12b-it

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

Duration: 49s

Actual run cost: \$0.000510

Projected execution cost: \$0.5610/mo (50/biz day × 22 days)

Report date: July 13, 2026 - 10:34 PM EST

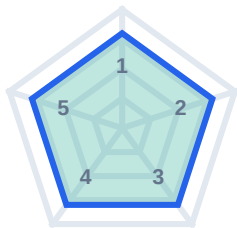
Overall

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: 13854 chars - Latency: 49s - 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: 49s

Engineering counterpart

2:33 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

2:33 AM - Responded in 11s

Okay, here's my triage and restoration plan for the `analytics.fct\_orders` mart freshness SLA breach, based on the provided metadata.

1. Root Cause Identification: Blocking Upstream Model

The Blocking Upstream Model: `intermediate.int\_order\_enriched` is the primary blocker. Its `last\_run\_utc` matches the last successful run of the mart, and its `status` is `failed` with a `DBT\_RUN\_TIMEOUT` error. This directly prevents `fct\_orders` from refreshing. While `staging.stg\_orders` and `staging.stg\_customers` are successful, they feed into `intermediate.int\_order\_enriched`, and its failure cascades downstream.

2. Blast Radius - Downstream Impact

Affected Consumers: The breach impacts two key consumers:

Looker Revenue Overview: This is a P1 dashboard, meaning decisions directly tied to revenue are being made with stale data. This is high-severity.

exec_daily_orders Slack bot: While likely less critical than the Looker dashboard, it still delivers daily order information to executives. Stale data introduces potential inaccuracies in their understanding.

3. Restoration Plan - Ordered Steps to Restore Mart within SLA

This plan prioritizes immediate recovery and then investigates the root cause of the timeout.

Phase 1: Immediate Recovery (Focus: Get the Mart Refreshing)

1. (0-15 minutes) - Investigate `int\_order\_enriched` Failure:

Action: Immediately check the DBT run logs/execution details for `intermediate.int\_order\_enriched`. The error message `DBT\_RUN\_TIMEOUT` after 45m is crucial. We need to understand which model or test within `int\_order\_enriched` is timing out. Tools like DBT's Cloud UI or our internal monitoring dashboards should provide this.

Goal: Identify the specific model(s) contributing to the timeout.

2. (15-30 minutes) - Quick Fix (If Possible and Safe):

Action (Potential): Based on the DBT run log analysis, consider a quick fix. Examples:

If a specific model is known to be problematic: Temporarily increase the timeout limit for that specific model in DBT profiles or configuration. (Use with caution! Monitor closely.)

If the timeout is related to data volume: If the model has a `LIMIT` clause that can be safely removed or increased, try that.

Goal: Attempt to unblock the `int\_order\_enriched` model and trigger a new run.

3. (30-45 minutes) - Trigger the Refresh:

Action: Manually trigger a refresh of `intermediate.int\_order\_enriched` and then `analytics.fct\_orders`. This can be done through our DBT deployment pipeline or a dedicated refresh tool.

Goal: Observe the refresh process closely. Monitor for successful completion (especially `int\_order\_enriched`) and check the freshness status of `fct\_orders`.

Phase 2: Root Cause Analysis & Prevention (Post-Recovery - Priority: High)

4. (45-60 minutes) - Verify SLA Compliance:

Action: Monitor the staleness of `fct\_orders` to ensure it falls back within the 120-minute SLA.

Goal: Confirm that the immediate recovery steps have resolved the freshness issue.

5. (60+ minutes) - Root Cause Investigation (Detailed)

Engineering counterpart

2:33 AM - Responded in 0s

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

Agent response

2:33 AM - Responded in 11s

Okay, let's break down the reasoning behind my response step-by-step, and outline the assumptions I made.

1. Initial Assessment & Identifying the Primary Issue

Step: I started by reading the entire JSON metadata to get a complete picture of the situation. I specifically looked for `status` fields, `last\_run\_utc`, and `error` messages.

Reasoning: Data engineering incident response is about quickly isolating the problem. The primary goal is to find what's preventing the mart from being fresh.

Finding: I immediately noticed the `analytics.fct\_orders` mart had a `status: BREACHED` and significant `staleness\_minutes` (388). The `last\_successful\_run\_utc` was significantly older than `now\_utc`.

2. Identifying the Blocking Upstream Model

Step: I examined the `upstream` array. Each entry represents a dependency. I looked for models with a `status: failed` or a significantly older `last\_run\_utc` than the mart itself.

Reasoning: A mart can only be refreshed if all its upstream dependencies are successful and relatively recent. A failed dependency directly blocks the downstream mart. The most recent execution time of an upstream model gives a good indication of how "fresh" the data is.

Finding: `intermediate.int\_order\_enriched` stood out. It had a `status: failed` and a `last\_run\_utc` that matched the `last\_successful\_run\_utc` of `analytics.fct\_orders`. This strongly suggests that `fct\_orders` hasn't run since `int\_order\_enriched` last failed. The `error: "DBT\_RUN\_TIMEOUT after 45m"` was also a critical piece of information.

3. Determining Blast Radius (Downstream Impact)

Step: I looked at the `downstream\_consumers` array. This lists the systems and users relying on the mart.

Reasoning: Understanding who's affected by stale data helps prioritize the response and communicate the impact.

Finding: "Looker Revenue Overview" and "exec_daily_orders Slack bot" were identified as downstream consumers. Given the mart name includes "orders" and the mention of "revenue," I assumed the Looker dashboard is critical for business decision-making (hence the P1 designation).

4. Developing the Restoration Plan - Prioritization & Steps

Step: I created a phased plan, prioritizing immediate recovery and then deeper investigation.

Reasoning: Incident response often involves a "break glass" approach to get things working quickly, followed by a longer-term investigation to prevent recurrence.

Phase 1 - Immediate Recovery:

Step 1: Investigate `int\_order\_enriched` Failure: The error message pointed to a timeout. I assumed that the DBT run logs would provide details on which specific model or test was causing the timeout.

Step 2: Quick Fix (If Possible): I considered a temporary timeout increase as a potentially fast solution, but acknowledged the risk of masking an underlying data issue. This assumes we have a mechanism to adjust timeouts quickly and safely.

Step 3: Trigger the Refresh: Manual triggering was proposed to bypass any automated scheduling issues.

Phase 2 - Root Cause Analysis & Prevention:

Step 4: Verify SLA Compliance: Ensuring the mart meets the SLA is the ultimate goal.

Step 5: Root Cause Investigation: A deeper analysis of the timeout is needed to

Engineering counterpart

2:33 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

2:34 AM - Responded in 14s

Okay, excellent question. Before shipping this restoration plan to production, I'd want to verify several things to minimize risk and ensure the plan's effectiveness. Here's a breakdown of what I'd verify, categorized by importance (Critical, Important, Nice-to-Have), and roughly the effort level required for each.

I. Critical Verifications (Must-Do - High Effort)

1. DBT Run Log Access & Comprehension: (Effort: 30-60 mins)

Verification: Absolutely confirm I can actually access the detailed DBT run logs for ``intermediate.int_order_enriched``. This is the single most critical point. Without access, the "Quick Fix" and Root Cause Investigations are dead in the water.

Verification: Confirm I understand the DBT run logs. Can I identify the specific model or test that's timing out? This often requires familiarity with the DBT project structure and the models involved.

2. Timeout Adjustment Mechanism: (Effort: 15-30 mins)

Verification: If the "Quick Fix" of adjusting the timeout is considered, I need to ensure that we have a safe and reliable way to do this. This means understanding:

Where timeouts are configured (DBT profiles, a central configuration repository, etc.).

The process for updating them (e.g., direct modification, a deployment pipeline).

The potential side effects of increasing the timeout (e.g., increased resource usage, masking a deeper problem).

Ideally, we have a process to temporarily overwrite environment variables to adjust the timeout, not modify a persistent configuration.

3. Refresh Trigger Mechanism: (Effort: 15 mins)

Verification: Confirm I know how to manually trigger a refresh of both ``intermediate.int_order_enriched`` and ``analytics.fct_orders``. Test this on a non-production environment if possible. Ensure I have the necessary permissions.

4. Rollback Plan: (Effort: 15 mins)

Verification: Write down a detailed rollback plan in case the quick fix doesn't work or makes the situation worse. For example, reverting timeout changes, and reverting to the last known good state of the mart.

II. Important Verifications (Should-Do - Medium Effort)

5. Downstream Consumer Alerting: (Effort: 15-30 mins)

Verification: Verify that the "Looker Revenue Overview" and "exec_daily_orders Slack bot" are properly configured to alert when the ``analytics.fct_orders`` mart is stale. This ensures we're notified if the issue recurs.

6. Data Volume Expectations: (Effort: 15 mins)

Verification: Get a rough understanding of the data volume processed by the models involved in ``intermediate.int_order_enriched``. A sudden spike in data could be the root cause of the timeout.

7. SLA Monitoring: (Effort: 10 mins)

Verification: Ensure we have proper monitoring in place to track the freshness SLA for ``analytics.fct_orders`` and that alerts are triggered promptly when breaches occur.

III. Nice-to-Have Verifications (Good to Know - Low Effort)

8. Recent Code Changes: (Effort: 5-10 mins)

Verification: Review recent code changes to ``intermediate.int_order_enriched`` or related models. A recent deployment could have introduced the timeout issue.

9. Resource Utilization: (Effort: 1

Engineering counterpart

2:34 AM - Responded in 0s

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

Agent response

2:34 AM - Responded in 12s

Okay, let's outline the top risks associated with this restoration plan and detail the rollback plan.

I. Top Risks

Here's a breakdown of the top risks, categorized by severity (High, Medium, Low), and with mitigation strategies included.

| Risk | Severity | Description | Mitigation |
|------|----------|-------------|------------|
|------|----------|-------------|------------|

| | | | |
|-------------------------------|------|---|--|
| Incorrect Timeout Adjustment: | High | Increasing the timeout doesn't actually fix the underlying problem (e.g., inefficient query, resource contention) and just masks it, leading to performance degradation or even more severe issues later. It might also allow a genuinely problematic query to consume excessive resources. | Thoroughly investigate the DBT run logs before adjusting the timeout. Monitor resource utilization (CPU, memory) closely after the change. Set a short-term expiry on the increased timeout. |
|-------------------------------|------|---|--|

| | | | |
|---------------------------|------|--|--|
| Data Quality Degradation: | High | The timeout issue is a symptom of a deeper data quality problem (e.g., corrupted data, unexpected data patterns). Increasing the timeout allows the faulty process to complete, potentially propagating incorrect data downstream. | While addressing the timeout, also implement data quality checks within <code>intermediate.int_order_enriched`</code> (e.g., validating data types, checking for null values, enforcing business rules). |
|---------------------------|------|--|--|

| | | | |
|-------------------------------|--------|--|--|
| Unforeseen Dependency Issues: | Medium | Adjusting the timeout in one model could have unexpected downstream effects on other models that depend on its output. | Test the timeout change in a non-production environment (if feasible) before applying it to production. Monitor dependent models closely after the change. |
|-------------------------------|--------|--|--|

| | | | |
|-------------------------|--------|---|---|
| Manual Refresh Failure: | Medium | The manual refresh process fails due to permissions issues, configuration errors, or transient system problems. | Double-check permissions and configuration before triggering the refresh. Have a backup plan for triggering the refresh (e.g., a different user account, a different tool). |
|-------------------------|--------|---|---|

| | | | |
|--------------------------|--------|--|--|
| SLA Breach Prolongation: | Medium | Despite the efforts, the mart remains stale, prolonging the SLA breach and impacting downstream consumers. | Maintain clear communication with downstream consumers about the ongoing issue and estimated time of resolution. Provide interim data (if available) to minimize the impact. |
|--------------------------|--------|--|--|

| | | | |
|---------------------------------|-----|---|---|
| Resource Contention Escalation: | Low | Increasing the timeout for one model exacerbates resource contention across the entire system, impacting other processes. | Monitor overall system resource utilization closely after the change. If contention increases, consider alternative solutions (e.g., optimizing queries, increasing compute resources). |
|---------------------------------|-----|---|---|

II. Rollback Plan

This plan outlines the steps to revert the changes if the restoration efforts fail, ensuring we return to a known stable state.

Trigger for Rollback:

- The mart remains stale after a reasonable amount of time (e.g., 30 minutes) following the timeout adjustment and refresh attempt.

- The downstream consumers report further issues or data inconsistencies.

- The system exhibits signs of increased instability or resource contention.

Rollback Steps (Prioritized):

1. (Immediate - within 5 mins): Revert Timeout Change:

Action: Immediately revert any changes made to the timeout configuration for `intermediate.int_order_enriched``. Restore it to the previously known good value. This is the highest priority step.

Method: Use the documented method for reverting the timeout change (e.g., reverting the environment variable override, reverting the configuration file).

2. (Within 10 mins): Stop Manual Refresh Attempts:

Action: Cease any further manual refresh attempts of `intermediate.int_order_enriched`` and `analytics.fct_orders``. Repeated manual interventions without resolving the root cause can make the situation worse.

3. (Within 15

Engineering counterpart

2:34 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.