



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/gemini-2.5-flash

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

Duration: 23s

Actual run cost: \$0.005300

Projected execution cost: \$5.83/mo (50/biz day × 22 days)

Report date: July 13, 2026 - 10:33 PM EST

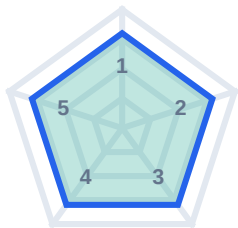
Overall

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: 12887 chars - Latency: 23s - 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: 23s

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 5s

Here's a triage report for the `analytics.fct\_orders` freshness SLA breach:

Freshness SLA Breach Triage: `analytics.fct\_orders`

Status: BREACHED (P1)

Current Staleness: 388 minutes

SLA: 120 minutes

Last Successful Refresh: 2026-07-05T09:12:00Z

Current Time: 2026-07-05T15:40:00Z

1. Upstream Model Blocking Refresh:

The upstream model blocking the refresh of `analytics.fct\_orders` and causing the SLA breach is:

 `intermediate.int\_order\_enriched`

 Last Run: 2026-07-05T09:12:00Z

 Status: `failed`

 Error: `DBT\_RUN\_TIMEOUT after 45m`

The `note` explicitly states, "int_order_enriched has been failing since morning; fct_orders has not refreshed," confirming this is the root cause.

2. Blast Radius on Downstream Consumers:

The following downstream consumers are currently receiving stale data due to this breach:

 Looker Revenue Overview: This is a P1 dashboard, indicating a significant impact on revenue visibility, reporting, and potentially decision-making. Stakeholders relying on this dashboard are currently looking at data that is over 6 hours old.

 exec_daily_orders Slack bot: This bot is likely delivering outdated order information to executives, which could lead to misinformed daily updates or actions.

Given the P1 tier, this breach has a high impact on critical business operations and executive reporting.

3. Ordered Steps to Restore the Mart Within SLA:

Here are the ordered steps to restore `analytics.fct\_orders` within its SLA:

Phase 1: Immediate Remediation (Restore Freshness)

1. Investigate `intermediate.int\_order\_enriched` Failure (High Priority):

 Access the logs for the `intermediate.int\_order\_enriched` DBT run that failed at 2026-07-05T09:12:00Z.

 Identify the exact SQL query or transformation step that is causing the `DBT\_RUN\_TIMEOUT`.

 Hypothesis: This could be due to:

 An inefficient query (e.g., missing indexes, complex joins, full table scans on large tables).

 Increased data volume leading to longer processing times.

 Resource contention on the data warehouse.

A data quality issue causing an infinite loop or extremely long computation.

A recent code change (if any) to ``int_order_enriched`` that introduced performance degradation.

2. Attempt Quick Fix / Rerun ``intermediate.int_order_enriched``:

If the cause is immediately apparent and minor (e.g., a simple syntax error fixed by a previous commit, or a known transient resource issue): Apply the fix and manually trigger a rerun of ``intermediate.int_order_enriched``.

If the cause is a performance bottleneck and cannot be immediately optimized for the current run: Consider a temporary workaround if feasible and safe, such as:

Materialization change: If it's currently a view, can it be temporarily materialized as a table for better performance?

(Unlikely to be a quick fix if it's already timing

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 5s

Okay, let's break down the reasoning behind each part of the triage, along with the assumptions made.

Reasoning Breakdown & Assumptions:

Overall Approach:

My approach follows a standard incident response methodology: Identify the problem, understand its impact, and then formulate a plan for resolution and prevention. The structure of the request (blocking upstream, blast radius, steps to restore) directly guides this.

1. Upstream Model Blocking Refresh:

Reasoning:

1. Identify the Target: The core problem is that `analytics.fct\_orders` is stale.
2. Check `fct\_orders`'s Own Status: The `freshness.last\_successful\_run\_utc` for `fct\_orders` is `2026-07-05T09:12:00Z`. This perfectly matches the `last\_run\_utc` of `intermediate.int\_order\_enriched` and its `failed` status. This is a strong indicator that `fct\_orders` depends on `int\_order\_enriched` and hasn't been able to run successfully since `int\_order\_enriched` failed.
3. Evaluate Other Upstreams:
 `staging.stg\_orders` and `staging.stg\_customers` both have recent `last\_run\_utc` times (`15:05Z` and `14:58Z` respectively) and `status: "success"`. This tells me they are not the problem; they are refreshing as expected.
4. Confirm with `note`: The `note` explicitly states: "`int\_order\_enriched` has been failing since morning; `fct\_orders` has not refreshed." This directly confirms the conclusion that `int\_order\_enriched` is the blocking upstream.
5. Identify the specific error: The `error: "DBT\_RUN\_TIMEOUT after 45m"` directly tells us why `int\_order\_enriched` is failing.

Assumptions Made:

Direct Dependency: I assume that `analytics.fct\_orders` directly depends on `intermediate.int\_order\_enriched`. This is a very safe assumption given the provided data, where `fct\_orders`'s last successful run aligns precisely with `int\_order\_enriched`'s last failed run.

Standard ETL/ELT Orchestration: I assume a typical data pipeline orchestration where a downstream model (`fct\_orders`) will not run if an upstream dependency (`int\_order\_enriched`) has failed or not completed successfully.

"Blocking" Implies "Must Succeed": The term "blocking" implies that the successful completion of `int\_order\_enriched` is a prerequisite for `fct\_orders` to run.

2. Blast Radius on Downstream Consumers:

Reasoning:

1. Identify Direct Consumers: The `downstream\_consumers` list explicitly names "Looker Revenue Overview" and "exec_daily_orders Slack bot".
2. Severity of `fct\_orders`: The `sla.tier` is "P1 revenue dashboard". This immediately flags `fct\_orders` as a critical asset, and any consumer of it is therefore also impacted by a critical issue.
3. Nature of Consumers:

Looker Revenue Overview: "Revenue Overview" implies high-level, business-critical financial reporting. "P1" corroborates this. Stale data here directly impacts financial analysis, performance tracking, and strategic decision-making.

exec_daily_orders Slack bot: "exec_daily

What would you verify before shipping this to production?

Agent response

2:33 AM - Responded in 6s

Before "shipping" this restored `analytics.fct\_orders` to production (i.e., declaring the incident resolved and moving on), I would perform the following verifications:

1. Verification of `intermediate.int\_order\_enriched` (Root Cause Fix):

Successful Completion: Confirm that `intermediate.int\_order\_enriched` has completed successfully without errors.

How: Check orchestration logs, DBT run status, or data warehouse `INFORMATION\_SCHEMA` for the last run of this model.

Performance Baseline: Compare the run time of the fixed `int\_order\_enriched` against its historical baseline.

Why: To ensure the fix hasn't merely shifted the timeout window (e.g., it now runs for 40 minutes instead of 45, but is still very slow) or introduced new performance issues.

How: Metrics from the orchestration tool (e.g., Airflow, dbt Cloud, Dagster).

Data Quality (Spot Check): Perform some quick data checks on `int\_order\_enriched`.

Why: To ensure the fix didn't introduce data corruption or unexpected changes. A timeout could mask underlying data issues.

How:

Row count comparison to previous successful runs.

Check for `NULL`s in critical columns that shouldn't be null.

Distribution checks on key numeric fields (e.g., sum of order amounts).

Comparison of a few sample records before and after the fix (if data is easily comparable).

2. Verification of `analytics.fct\_orders` (Impacted Mart):

Successful Completion: Confirm that `analytics.fct\_orders` has completed successfully.

How: Check orchestration logs, DBT run status, or data warehouse `INFORMATION\_SCHEMA` for the last run of this model.

Freshness within SLA: Verify that `fct\_orders`'s `last\_successful\_run\_utc` is now recent enough to be within the 120-minute SLA.

How: Calculate `now\_utc - last\_successful\_run\_utc` and ensure it's < 120 minutes.

Data Quality (Spot Check): Perform essential data quality checks on `analytics.fct\_orders`. This is critical for downstream consumers.

Why: Even if `int\_order\_enriched` is fixed, an issue in `fct\_orders`'s own logic could still cause problems.

How:

Row Count: Compare total row count to previous successful runs. Look for significant deviations.

Key Metrics: Check aggregates like total revenue, number of unique orders/customers for the latest refresh period and compare against expectations (e.g., same day last week/month, historical trends).

Nulls/Duplicates: Check for `NULL`s in primary keys or other critical columns, and ensure no unexpected duplicates.

Recent Data: Confirm that the latest data from the source systems (e.g., today's orders) is present in `fct\_orders`.

Data Tests: If automated data tests exist for `fct\_orders` (e.g., dbt tests), ensure they all pass.

3. Verification of Downstream Consumers:

Looker Revenue Overview:

Visual Confirmation: Access the dashboard and visually confirm that the data appears fresh and correct for the latest period.

Refresh Check: Confirm that the dashboard itself

Engineering counterpart
2:33 AM - Responded in 0s

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

Agent response

2:33 AM - Responded in 7s

Okay, great questions. Understanding the risks and having a rollback plan is crucial before implementing any fix.

Top Risks:

1. Re-introducing the Timeout/Performance Issue:

Risk: The fix for ``int_order_enriched`` might be incomplete, incorrect, or only a temporary band-aid. The model could time out again during the next run, or even during the current manual rerun, putting us back to square one or delaying the overall resolution.

Mitigation during fix: Thoroughly test the proposed fix in a staging environment if possible. Analyze query plans, resource usage, and data volume changes.

Mitigation after fix (verification): Closely monitor the run time and resource consumption of ``int_order_enriched`` post-fix.

2. Data Quality Issues / Data Corruption:

Risk: The fix for ``int_order_enriched`` (e.g., query rewrite, index change, temporary data filtering) might inadvertently introduce incorrect calculations, missing data, duplicate records, or other data quality problems. This could then propagate to ``fct_orders`` and subsequently to critical dashboards and reports.

Mitigation: Rigorous data quality checks (row counts, sum checks, null checks, uniqueness checks for PKs) on both ``int_order_enriched`` and ``fct_orders`` after the fix. Automated data tests (dbt tests) are invaluable here.

3. New Production Incident / Unforeseen Side Effects:

Risk: Any change to a core intermediate model can have ripple effects throughout the data warehouse that are difficult to predict. A seemingly benign change might interact poorly with other models, views, or data consumers, leading to new failures or performance degradation elsewhere.

Mitigation: Understand the full dependency graph of ``int_order_enriched`` (beyond just ``fct_orders``). Communicate changes widely within the team. Monitor overall data platform health and error logs closely for a period after the fix.

4. Inability to Meet SLA Despite Fix:

Risk: Even if the fix for ``int_order_enriched`` works, the accumulated backlog or the sheer time it takes to run both ``int_order_enriched`` and ``fct_orders`` might mean we still can't get ``fct_orders`` within its 120-minute SLA for the current cycle.

Mitigation: Prioritize expediting the runs. Be realistic about achievable recovery time. Communicate potential continued SLA breach proactively to stakeholders if this seems likely.

5. Lack of Understanding of Root Cause:

Risk: A quick fix (e.g., simply rerunning, or a superficial change) might resolve the immediate timeout but doesn't address the underlying systemic issue (e.g., steadily growing data, inefficient architecture, unsustainable query patterns). This leads to repeat incidents.

Mitigation: Dedicate time after the immediate incident resolution to a deeper dive into the root cause. This is part of the "Next Steps / Prevention" phase.

Rollback Plan:

The nature of a timeout during a DBT run makes a traditional "rollback" complex, as it's not a deployment of new code that can be reverted, but rather a failure to execute existing code. However, we can define a rollback strategy for the attempts to fix the issue.

The primary goal of a rollback here is to revert any changes made during the attempted fix and return the system to its previous stable (though breached) state, allowing for a re-evaluation of the approach or escalation.

Scenario: We attempt a fix (e.g.,

Engineering counterpart

2:33 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.