



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: openai/gpt-4.1-nano

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

Duration: 16s

Actual run cost: \$0.001160

Projected execution cost: \$1.28/mo (50/biz day × 22 days)

Report date: July 14, 2026 - 2:55 AM EST

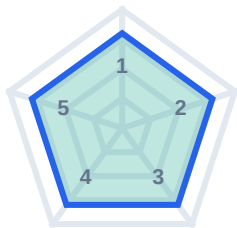
Overall

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: 11001 chars - Latency: 16s - 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: 16s

Engineering counterpart

6:55 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

6:55 AM - Responded in 4s

The root cause of the freshness SLA breach is a failure in the upstream model ``intermediate.int_order_enriched`` due to a `DBT_RUN_TIMEOUT` error, which has persisted since the morning. This model is critical because it feeds into the ``analytics.fct_orders`` mart, which is now stale by approximately 6 hours (388 minutes vs. SLA of 120 minutes).

Impact Analysis

- Blocking upstream model:

``intermediate.int_order_enriched`` is the primary blocker preventing ``analytics.fct_orders`` from refreshing. Since downstream consumers rely on ``fct_orders``, they are also impacted.

- Blast radius:

- Primary: ``Looker Revenue Overview`` dashboard, which depends on ``fct_orders`` for revenue metrics.
- Secondary: ``exec_daily_orders`` Slack bot, which likely reports or triggers alerts based on order data.

Ordered Steps to Restore Freshness Within SLA

1. Investigate the failure of ``intermediate.int_order_enriched``:

- Check logs and error details for the timeout.
- Identify if the timeout was due to a query issue, resource contention, or external dependencies.

2. Attempt a manual rerun of ``intermediate.int_order_enriched``:

- If the issue was transient (e.g., resource spike), rerun the model manually.
- Monitor for successful completion.

3. If the rerun fails again:

- Review and optimize the model's query performance or resource allocation.
- Consider increasing timeout settings temporarily to allow completion if resource constraints are causing timeout.

4. Once ``intermediate.int_order_enriched`` succeeds:

- Trigger a refresh of ``analytics.fct_orders``.
- Confirm the latest successful run timestamp and ensure it is within SLA (less than 120 minutes stale).

5. Validate downstream consumers:

- Confirm that ``Looker Revenue Overview`` dashboard and ``exec_daily_orders`` bot are updated with fresh data.

6. Communicate the issue resolution:

- Send an update to stakeholders with confirmation of restored freshness.
- Document the incident and root cause for future reference.

Summary

- Blocking upstream model: ``intermediate.int_order_enriched``
- Downstream blast radius: ``Looker Revenue Overview``, ``exec_daily_orders``
- Restoration plan: Investigate, rerun, optimize, then refresh ``fct_orders`` and downstream consumers within SLA.

Let me know if you'd like detailed logs or further assistance with rerun commands.

Engineering counterpart

6:55 AM - Responded in 0s

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

Agent response

6:55 AM - Responded in 5s

Certainly. Here's my step-by-step reasoning process, along with the assumptions I made:

1. Identify the root cause of the SLA breach

- Observation:

The metadata shows `fct\_orders` last successfully refreshed at 2026-07-05T09:12:00Z, but current time is 15:40 UTC, making it over 6 hours stale (388 minutes), breaching the SLA of 120 minutes.

- Freshness status:

The note indicates that `int\_order\_enriched` has been failing since the morning, with a `DBT\_RUN\_TIMEOUT` error.

- Assumption:

The failure of `intermediate.int\_order\_enriched` is the primary blocker preventing `fct\_orders` refresh, since `fct\_orders` depends on this upstream model.

This assumption is based on the typical data pipeline flow where downstream tables cannot be refreshed until upstream models succeed.

2. Determine whether upstream models other than the failed one are contributing to the problem

- Upstream models:

- `staging.stg\_orders` and `staging.stg\_customers` show success with recent run timestamps.

- Assumption:

Since these models are successful and recent, they are not the bottleneck. The failure in `intermediate.int\_order\_enriched` is isolated and causing the entire refresh blockage.

3. Assess impact to downstream consumers

- Impact:

`fct\_orders` is the direct source for downstream consumers like the Looker dashboard and Slack bot.

- Assumption:

These downstream consumers are dependent on `fct\_orders` and are now stale, which justifies prioritizing restoring `fct\_orders`.

4. Develop a corrective action plan

- Step 1: Investigate the failure of the critical model `intermediate.int\_order\_enriched` - understanding the cause (`DBT\_RUN\_TIMEOUT`).

- Step 2: Attempt a manual rerun, assuming transient issues may have caused the timeout.

- Assumption:

Rerunning the model might succeed if the underlying issue was temporary (e.g., resource spike).

- Step 3: If reruns fail again, consider optimizing the model or increasing timeout limits, assuming the problem may be related to query complexity or resource constraints.

- Step 4: Once the upstream model succeeds, trigger a refresh of `fct\_orders`, assuming it will now succeed.

5. Validate and communicate resolution

- Assumption:

After refreshing `fct\_orders`, downstream consumers will automatically update, or can be manually refreshed if needed.

6. Additional assumptions and caveats

- The `DBT\_RUN\_TIMEOUT` suggests a query or resource issue rather than data corruption or logic errors, so rerunning might resolve it.
- I assume no other models or external dependencies are causing delays, based on the provided metadata.
- I assume that rerunning models manually is feasible within operational constraints.

Summary of assumptions:

- `intermediate.int\_order\_enriched` is the root cause blocking downstream refresh.
- The failure is transient (`DBT\_RUN\_TIMEOUT`) and can be remedied via rerun.
- Other upstream models are healthy.
- Downstream consumers depend on `fct\_orders` for their data.
- Rerunning models and refreshing downstreams will restore within SLA.

Please let me know if you'd like to explore any of these steps in more detail or verify assumptions.

Engineering counterpart
6:55 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

6:55 AM - Responded in 3s

Before shipping the resolution steps to production, I would verify the following to ensure stability, correctness, and minimal risk:

1. Root Cause Confirmation

- Check logs and error details for ``intermediate.int_order_enriched`` reruns to confirm the timeout was transient and not due to a systemic issue (e.g., query logic, data volume spikes, external dependencies).
- Validate that the timeout was due to a temporary resource constraint rather than a persistent bug, logic error, or data corruption.

2. Model Performance and Resource Usage

- Assess the query plan and resource consumption of ``intermediate.int_order_enriched`` to identify potential bottlenecks.
- Implement optimizations or index tuning if applicable, to reduce timeout likelihood.
- Review recent changes in the model or underlying data that could have caused increased runtime.

3. Operational Readiness

- Ensure rerun mechanisms are reliable (e.g., via Airflow, dbt Cloud, or orchestration system).
- Verify rerun permissions and credentials are intact.
- Confirm that rerunning the model won't cause data inconsistencies or partial loads.

4. Testing the Fix

- Perform a dry run or test rerun in a staging or non-production environment if possible.
- Monitor the rerun's success and runtime.

5. Post-Rerun Validation

- After rerun, verify that ``intermediate.int_order_enriched`` completes successfully.
- Confirm that ``fct_orders`` refreshes correctly and contains recent, consistent data.
- Check downstream consumers (Looker, Slack bot) for updated, correct data.

6. Contingency Planning

- Prepare rollback procedures in case the rerun causes unexpected issues.
- Set up alerting or monitoring for subsequent failures or slowdowns.

7. Communication

- Ensure stakeholders are informed of the fix, especially if this involves manual reruns or temporary configuration changes.
- Document the incident, root cause, and resolution steps for future incidents.

Summary

Verification is about confirming that the root cause is truly transient, the model performance is stable, reruns are safe, and downstream data integrity is maintained before deploying the fix into production. This minimizes the risk of recurrence and ensures consistent data freshness.

Engineering counterpart
6:55 AM - Responded in 0s

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

Agent response

6:55 AM - Responded in 4s

Top Risks

1. Transient Issue Reoccurs or Worsens

- The timeout may have been caused by an underlying persistent problem such as increased data volume, inefficient query logic, or resource contention.
- Rerunning the model again might fail or take longer, further delaying data availability or causing additional failures.

2. Data Inconsistency or Partial Loads

- Manual reruns or quick fixes might lead to partial, inconsistent, or stale data in the ``intermediate.int_order_enriched`` model or downstream tables.
- This can compromise downstream analytics or dashboards, leading to incorrect decision-making.

3. Resource Exhaustion or System Instability

- Rerunning large or complex models might spike resource usage, impacting other jobs or causing system instability.
- If resource constraints are not addressed, subsequent reruns may also fail.

4. Operational Risks

- Reruns outside of controlled environments or without proper validation could introduce data anomalies.
- Manual intervention might bypass built-in safeguards, leading to unintended side effects.

5. Stakeholder Impact

- Even if the rerun is successful, delays could impact critical reports or dashboards, affecting business operations.

Rollback Plan if the Fix Doesn't Work

1. Revert to the Last Successful State

- Restore the ``fct_orders`` and ``intermediate.int_order_enriched`` models to the last known successful build or snapshot.
- Use backups, versioned models, or previous stable states if available.

2. Pause or Disable the Failing Model

- Temporarily disable or comment out the problematic model to prevent it from re-executing until the root cause is addressed.
- Ensure downstream dependencies are aware to avoid propagating stale or partial data.

3. Notify Stakeholders and Update Monitoring

- Communicate the issue and the rollback status.
- Increase monitoring on the fallback data to identify any data gaps or issues.

4. Conduct a Root Cause Analysis

- Engage the data engineering team to identify why the rerun failed or caused issues.
- Determine whether the problem was a resource, logic, or external factor.

5. Plan for Reconciliation

- Once root causes are addressed, plan a controlled re-execution with staged validation.
- Implement safeguards (e.g., smaller incremental runs, increased timeouts, resource scaling).

6. Document Lessons Learned

- Record the incident, risks encountered, and the steps taken for future reference and process improvement.

Summary:

The primary risk is that rerunning the model may not resolve the issue or could introduce data inconsistencies. The rollback plan involves reverting to the last good state, disabling the problematic model, communicating transparently, and thoroughly investigating root causes to prevent recurrence.

Engineering counterpart

6:55 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.