



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: deepseek/deepseek-v4-flash

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

Duration: 2m 23s

Actual run cost: \$0.001232

Projected execution cost: \$1.36/mo (50/biz day × 22 days)

Report date: July 14, 2026 - 2:54 AM EST

Overall

8.0/10

PASS - trust=high

Pass - score 8.0/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



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: 9328 chars - Latency: 143s - 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: 2m 23s

Engineering counterpart

6:52 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

6:53 AM - Responded in 48s

Upstream blocker:

`intermediate.int\_order\_enriched` - failed at 09:12 UTC with `DBT\_RUN\_TIMEOUT` after 45m`. This model is the only failing dependency; `fct_orders` cannot refresh until it completes successfully.

Blast radius on downstream consumers:

- Looker Revenue Overview - shows stale data (last refreshed ~09:12 UTC).

- exec_daily_orders Slack bot - will not fire or will send outdated aggregates.

Both are P1 revenue dashboards/alerts; the breach affects revenue visibility and operational decisions.

Ordered steps to restore `fct\_orders` within SLA (target: refreshed by 17:40 UTC, 120 min from now):

1. Examine the timeout root cause - Check the `int\_order\_enriched` run log (e.g., DBT artifact, Snowflake query history) for:

- Long?running SQL (e.g., missing join keys, full table scan, unoptimized window function).

- Warehouse resource contention (e.g., oversized concurrent queries, XS warehouse overwhelmed).

2. Apply a quick fix - Based on findings:

- If the SQL is inefficient: add a `LIMIT 1000` or temporary filter to reduce data volume, OR rewrite the query (e.g., add incremental logic, change join order).

- If timeout is resource?related: temporarily scale up the warehouse (e.g., XS -> S) or kill other queries.

- If the timeout

Engineering counterpart

6:53 AM - Responded in 0s

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

Agent response

6:54 AM - Responded in 55s

Thanks for asking me to unpack that - it's a good way to catch hidden assumptions and validate the plan. Let me walk you through my thought process step by step, and then I'll state the assumptions I made explicitly.

1. How I identified the blocker

Step 1 - Compare freshness timestamps.

The mart's `last\_successful\_run\_utc` is `09:12`.

`stg\_orders` and `stg\_customers` ran much later (~15:00), so they are not blocking.

`int\_order\_enriched` has `last\_run\_utc: 09:12` and status `failed` - that matches the mart's last run time exactly.

Step 2 - Infer the dependency chain.

In a typical dbt DAG, a mart (like `fct\_orders`) sits at the end of a pipeline: staging -> intermediate -> mart. The note says `fct\_orders` has not refreshed, and `int\_order\_enriched` is the only upstream model that failed. So the mart cannot refresh until the intermediate model succeeds. No other upstream model shows a blocker.

Step 3 - Read the error message.

`DBT\_RUN\_TIMEOUT` after 45m - this is a resource or performance issue, not a syntax error. It suggests the model attempted to run for 45 minutes and was killed (likely by a timeout limit in dbt or the warehouse).

Assumption rooted here: The dependency chain is linear for this branch - `fct\_orders` depends directly on `int\_order\_enriched`, and `int\_order\_enriched` depends only on the staging models (which passed). That's typical but not proven. If the mart had other failing dependencies, my answer would change.

2. How I determined the blast radius

Step 1 - List downstream consumers from the JSON.

They are: `Looker Revenue Overview` and `exec\_daily\_orders Slack bot`.

Step 2 - Judge impact from the SLA tier.

The SLA is "P1 revenue dashboard" - so these are executive-facing, revenue-critical surfaces. Any staleness means decisions (e.g., pricing, forecasting) are based on data that is 6.5 hours old.

Step 3 - Consider what the consumers do.

- Looker dashboard: likely shows daily revenue figures. Stale for 388 minutes means the dashboard shows data from the morning, missing afternoon orders.

- Slack bot: probably posts a daily summary at a fixed time (e.g., 16:00 UTC). If it fires now, it will send old data - misleading.

Assumptions made:

- The Slack bot does not have its own freshness check or fallback to cached data.

- Looker does not have a separate caching layer that masks the staleness.

- The consumers are actively used - not dead dashboards.

- No other marts or views depend on `fct\_orders` (the JSON only listed those two, so I assumed they are the only downstream surfaces). If the blast radius were wider, my escalation steps might differ.

3. How I ordered the restoration steps

Step 1 - Recognize time pressure.

Current staleness: 388 min. SLA: 120 min. To meet SLA, I need the mart fresh by 17:40 UTC (15:40 + 2h). That's only 2 hours - so I must act quickly and avoid perfecting the root cause. A "quick fix" is the only way to restore within the window.

Step 2 - Decide the quickest path.

The timeout could be due to:

- A slow query (missing index, join explosion, unoptimized window

Engineering counterpart
6:54 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

6:54 AM - Responded in 30s

Great question - this is where we move from "triage under pressure" to "safe, sustainable fix." Before shipping the quick fix to production, I would verify the following in this order:

1. Verify the root cause diagnosis (not just symptoms)

What to check:

- Query history in Snowflake/BigQuery for the failed ``int_order_enriched`` run at 09:12 UTC - look at:
 - Actual execution time vs. the 45m timeout
 - Profile: is it spending time in ``COMPILE``, ``QUEUED``, or ``EXECUTION``?
 - Bytes scanned vs. bytes written - is it a full table scan on a large table?
- Concurrent workload during that run - was the warehouse overloaded by other jobs?
- dbt logs - confirm the error was indeed a timeout and not a SQL error that was misclassified

Why this matters: If I misdiagnose (e.g., assume it's a resource issue when it's actually a bad join that will produce wrong results), my "quick fix" could corrupt the mart.

2. Verify the quick fix works in a sandbox

What to do:

- Run the modified ``int_order_enriched`` SQL in a separate warehouse or dev schema with production data volume
- Confirm:
 - It completes within the timeout window (should run well under 45m)
 - Row count and key metrics match the last successful run from yesterday
 - No data quality anomalies (e.g., duplicate keys, nulls in unexpected places, cardinality changes)

Example check:

```
``sql
-- Compare row counts
SELECT COUNT() FROM dev.int_order_enriched WHERE run_date = CURRENT_DATE
UNION ALL
SELECT COUNT() FROM prod.int_order_enriched WHERE run_date = '2026-07-04';
``
```

3. Verify the downstream mart can refresh

What to do:

- Run ``fct_orders`` in the same dev environment using the new ``int_order_enriched``
- Check:
 - It completes within SLA window (ideally < 30 min for the full pipeline)
 - The output matches expected patterns (e.g., no missing date range, correct aggregation)

4. Verify the blast radius is contained

What to check:

- Confirm no other marts depend on ``int_order_enriched`` that weren't listed in the metadata (query the dbt manifest or data

catalog)

- If there are: coordinate their owners before the fix goes out (they may need to re-run after us)

5. Verify rollback readiness

What to do:

- Ensure the previous working version of `int\_order\_enriched` is tagged and deployable
- Confirm we can revert `fct\_orders` to the pre-breach state if the fix introduces a worse bug
- Have a `revert` button ready in the CI/CD pipeline

6. Verify the monitoring can catch regression

What to check:

- The SLA freshness check will re-run after deployment - confirm it alerts us if the fix doesn't actually improve freshness
- Set a temporary tighter alert (e.g., 30 min staleness) for the next 2 hours post-deploy

Key assumptions I'd validate explicitly

| Assumption I made earlier | How I'd verify it |

|---|---|

| `int\_order\_enriched` is the only blocker | Check the full DAG - are there any other upstream models that also failed but weren't listed? |

| The timeout is from SQL/resource, not data corruption | Look at the error logs - was it a `TIMEOUT` or a `MEMORY\_EXCEEDED` or something else? |

| The quick fix

Engineering counterpart
6:54 AM - Responded in 0s

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

Agent response

6:54 AM - Responded in 10s

Top Risks of the Proposed Fix

| Risk | Impact | Likelihood |

|-----|-----|-----|

| Quick fix doesn't resolve the timeout (e.g., added LIMIT still fails due to underlying data skew) | Mart remains stale; SLA breach widens | Medium |

| Fix introduces data quality issues (e.g., missing rows, wrong aggregations from a filter) | Looker dashboard shows incorrect revenue; executive decisions based on bad data | Medium-High |

| Fix works in dev but fails in production (different data volume, concurrent workload) | No improvement; wasted time | Medium |

| Downstream mart refresh fails (other dependencies not captured, e.g., a second intermediate model also timed out) | Partial refresh; data inconsistency | Low |

| Fix breaks downstream consumer schema (e.g., removed a column that Looker expects) | Dashboard breaks, alert fatigue | Low (if we test schema) |

| Time pressure leads to incomplete testing - we ship untested code | Every risk above amplifies | High |

Rollback Plan (if the fix fails)

Immediate actions (within 15 minutes of detecting failure)

1. Revert the code change

- `git revert <last\_commit>` or redeploy the previous version of `int\_order\_enriched` via CI/CD.
- Do not merge the fix into the main branch until thoroughly validated.

2. Restore the mart's data state

- If `fct\_orders` was already refreshed with bad data:
 - Option A: Restore from a known-good table snapshot (e.g., time-travel in Snowflake: `SELECT \* FROM analytics.fct\_orders AT(TIMESTAMP => '2026-07-05 09:12:00')`)
 - Option B

Engineering counterpart
6:54 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.