



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-oss-20b

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

Duration: 54s

Actual run cost: \$0.000362

Projected execution cost: \$0.3982/mo (50/biz day × 22 days)

Report date: July 14, 2026 - 2:56 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: 6941 chars - Latency: 54s - 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: 54s

Engineering counterpart

6:56 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

6:56 AM - Responded in 5s

? Freshness SLA Breach - Quick?Triage (P1)

| Item | Detail |

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

| Mart | `analytics.fct\_orders` |

| Owner | data?platform |

| SLA | 120?min max staleness (P1 revenue dashboard) |

| Current staleness | 388?min (??6?h?28?min) |

| Status | BREACHED |

| Upstream status | `stg\_orders` & `stg\_customers` succeeded (15:05?/?14:58 UTC).
 `int\_order\_enriched` failed (last_run 09:12?UTC, DBT_RUN_TIMEOUT after 45?min). |

| Downstream consumers | -?Looker - Revenue Overview
-?Slack - exec_daily_orders bot |

| Immediate note | `int\_order\_enriched` has been failing all morning; `fct\_orders` has never refreshed since then. |

> TL;DR

> The blocking model is `intermediate.int\_order\_enriched` - its failure stops `fct\_orders` from running.

> The impact is felt by the Revenue Overview dashboard and the daily Slack bot.

> The goal is to get a successful run of `int\_order\_enriched` within the next 30?min (so the mart can finish <?120?min stale).

1?? Identify & Fix the Blocking Model

| Step | Action | Owner | Target Time |

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

| 1. Pull DBT logs | `dbt run --select intermediate.int\_order\_enriched -profile data-platform -s` (only the failing model). Check the log that caused

Engineering counterpart

6:56 AM - Responded in 0s

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

Agent response

6:56 AM - Responded in 30s

? Step-by-Step Reasoning

| # | What I looked at | Why it matters | Assumptions |

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

| 1 | Freshness metadata (`last\_successful\_run\_utc`, `staleness\_minutes`, `status`) | Tells us why the SLA is breached and how long it is overdue. | Assume the timestamp is UTC and accurate. |

| 2 | Upstream models' status | If any upstream fails, downstream can't run. | Assume a direct dependency chain: `fct\_orders` -> `int\_order\_enriched` -> `stg\_orders`/`stg\_customers`. |

| 3 | Error on `int\_order\_enriched` (`DBT\_RUN\_TIMEOUT` after 45m) | Indicates the model is hanging / resource-hungry, blocking the mart. | Assume DBT timeout is the real reason, not a mis-reported status. |

| 4 | Downstream consumers | Determines blast radius and urgency. | Assume every consumer depends on the latest mart snapshot. |

| 5 | Note field | Confirms that the failure has persisted all morning. | Accept the note as reliable. |

| 6 | SLA tier (`P1` revenue dashboard) | Guides how aggressive we must be. | Assume P1 means 80%hour uptime in our org, 120min max staleness is the hard boundary. |

Conclusion of reasoning

The only upstream model that failed (`int\_order\_enriched`) is the direct parent of the mart.

Because that model failed, `fct\_orders` never ran after its last success at 09:12UTC.

The mart's staleness (388min) is well beyond the 120minute SLA, so the breach is P-level.

The two downstream consumers (Looker dashboard & Slack bot) will continue showing stale data until the mart refreshes.

? Blast Radius Analysis

| Consumer | Impact | Time to impact (if we ignore the stale mart) | Mitigation options |

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

| Looker - Revenue Overview | Data out-of-sync for 6+?h, could mislead executives | 6?h 28?min (current staleness) | 1) Display "Data Loading" banner
2) Show last-known-good snapshot |

| Slack - exec_daily_orders bot | Bot returns old totals, risk of wrong alerts | 6?h 28?min | 1) Suspend bot until refresh
2) Flag messages as "stale" |

Assumptions on impact

Both consumers query the mart directly (no caching layer that would mask staleness).

Users expect real-time revenue figures; a 6-hour lag is unacceptable for a P1 dashboard.

? Ordered Steps to Restore Within SLA

| # | Step | Action | Owner | Estimated time | Tool/Command |

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

| 1 | Validate DBT job state | Check dbt logs (`/var/log/dbt/...`) for the failing run. | Data-platform Ops | 5?min | `less /var/log

Engineering counterpart

6:56 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

6:56 AM - Responded in 8s

? What to Verify _before_ Shipping the Fix to Production

Below is a ready-to-use checklist you can hand-off to the Ops/QA team. Each item is grouped by the stage of the deployment cycle (Build -> Test -> Run -> Monitor -> Rollback). Check-off each line, document the result, and only then push the commit to `prod`.

| Stage | Verification | Why

Engineering counterpart

6:56 AM - Responded in 0s

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

Agent response

6:56 AM - Responded in 12s

?? Top Risks & Rollback Blueprint

| # | Risk | Why It's Critical | Mitigation?Check (pre?deploy) | Rollback Trigger | Rollback Action |

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

| 1 | Data corruption in `int\_order\_enriched` (e.g., wrong join logic, omitted fields) | The mart and downstream dashboards depend on accurate order?enrichment. A subtle change can silently drift revenue numbers. | - Unit?test the SQL with a sample dataset (hash of 100 rows).
- Compare `row\_count` & `checksum` against the last?good run. | If post?run `checksum` mismatch or manual audit shows incorrect totals. | - Re?deploy the last?good dbt commit (git tag vX.Y-good).
- Run `dbt snapshot` to restore table to previous state if snapshot exists. |

| 2 | Resource exhaustion / OOM / DB lock | The 45?min timeout hints at a runaway query. Restarting it could choke the database or lock other jobs. | - Profile the new query plan with `EXPLAIN (ANALYZE, BUFFERS)`.
- Verify worker/DB limits (CPU, memory, connection pool). | If query exceeds allowed time or triggers DB alerts. | - Revert to previous query version.
- Increase resource limits only after confirming no lock contention. |

| 3 | Dependency drift | New fields or changed semantics in `stg\_orders`/`stg\_customers` can break downstream consumers that were unaware. | - Run "schema diff" tool (e.g., `duckdb -e "SELECT FROM information\_schema.columns ...")`.
- Ensure Looker model files still reference existing columns. | If downstream errors (e.g., Looker "column not found"). | - Revert upstream models to the older schema commit.
- Update downstream models to use a safe?fallback alias. |

| 4 | SLA cascade | A failed refresh of `fct\_orders` may cascade to other marts that depend on it. | - Run a dependency graph audit (`dbt deps --graph`).
- Mark affected marts as "stale" in Ops dashboard. | If any dependent mart shows `staleness\_minutes > 120`. | - Stop the dependent dbt run.
- Deploy a minimal "stale data" mode that keeps old snapshot until the mart is refreshed. |

| 5 | Downtime for critical dashboards | Revenue decisions rely on near?real?time figures; a 6?hour lag could mislead executives. | - Put Looker dashboards in "maintenance" mode (display a banner).
- Disable the Slack bot or switch to "last?good" payload. | If SLA breach persists after 30?min of attempted recovery. | - Permanently disable the affected dashboards for the day.
- Notify stakeholders with an incident?report. |

? Rollback

Engineering counterpart

6:56 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.