



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: inclusionai/ling-2.6-flash

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

Duration: 15s

Actual run cost: \$0.000102

Projected execution cost: \$0.1122/mo (50/biz day × 22 days)

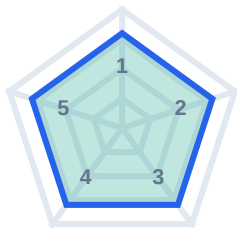
Report date: July 13, 2026 - 10:34 PM EST

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: 10022 chars - Latency: 15s - 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: 15s

Engineering counterpart

2:34 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

2:34 AM - Responded in 4s

Quick summary + immediate actions

- Root cause: `intermediate.int\_order\_enriched` has been failing since ~09:12 UTC (DBT_RUN_TIMEOUT).
- Blocking model: `intermediate.int\_order\_enriched` is the upstream blocking `fct\_orders` refresh.
- Blast radius: Downstream consumers (`Looker Revenue Overview`, `exec\_daily\_orders` Slack bot) are seeing stale data for ~6+ hours; P1 revenue dashboard impacted.
- Time to restore: Aim to restore `fct\_orders` within 60 minutes to get back inside the 120-minute SLA.

1) Which upstream model is blocking refresh?

`intermediate.int\_order\_enriched`

- Last successful run: `2026-07-05T09:12:00Z` (same time as `fct\_orders` breach).
- Status: failed with `DBT\_RUN\_TIMEOUT` after 45m.
- `fct\_orders` depends on `int\_order\_enriched`, so the mart refresh cannot proceed.

2) Blast radius on downstream consumers

- Looker Revenue Overview: Stale / potentially empty metrics since 09:12 UTC.
- exec_daily_orders Slack bot: Delivering outdated order counts / totals.
- Impact window: ~6.3 hours of missing fresh data (now 15:40 UTC vs last successful 09:12 UTC).
- Revenue decisions and executive reporting affected immediately.

3) Ordered steps to restore the mart within SLA

T+0-10 min - Stabilize & triage

- Confirm `staging.stg\_orders` and `staging.stg\_customers` are healthy (they are).
- Open incident channel; notify platform + revenue owners.

T+10-40 min - Diagnose & quick win

- Check if `int\_order\_enriched` failure is due to:
 - Resource saturation (warehouse/concurrency); try increasing warehouse size or cost limit.
 - Problematic joins / models; isolate to specific model or step.
- If it's a transient resource issue, retry `int\_order\_enriched` immediately.

T+40-70 min - Restore intermediate + mart

- If retry succeeds:
 - Run `fct\_orders` backfill (or trigger incremental) as soon as `int\_order\_enriched` completes.
- If retry fails:
 - Fallback to most recent good state of `int\_order\_enriched` (or last successful build artifacts).
 - Consider reverting `fct\_orders` to a previously successful run with a "data recency" disclaimer while longer fix is applied.

T+70-90 min - Validate & propagate

- Validate `fct\_orders` freshness and row counts.
- Trigger downstream consumers refresh (Looker, Slack bot).
- Confirm SLA recovered (< 120 min staleness).

T+90-120 min - Postmortem & safeguards

- Publish incident summary; add alerts on `int\_order\_enriched` run duration / failure rate.
- Implement auto-scaling / warehouse resize or materialization schedule to prevent recurrence.

- Add freshness SLAs & synthetic tests for `fct\_orders` and upstream critical models.

Decision point now: If `int\_order\_enriched` cannot be fixed in next 30-40 minutes

Engineering counterpart

2:34 AM - Responded in 0s

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

Agent response

2:34 AM - Responded in 4s

Here's my step-by-step reasoning and the assumptions behind it.

1) Parse the SLA and freshness data

- The P1 mart must be within 120 minutes of freshness.
- Now (15:40) ? last_successful_run (09:12) = 388 minutes stale -> SLA breached.
- Therefore we have at most 120 minutes from now to restore freshness; practically, we need fct_orders running again within ~60-90 minutes to have a safe buffer.

2) Identify the blocking upstream model

- Downstream models (staging.stg_orders and staging.stg_customers) succeeded recently and do not block fct_orders.
- intermediate.int_order_enriched failed at 09:12 with DBT_RUN_TIMEOUT and is an upstream dependency for fct_orders; therefore fct_orders cannot complete while int_order_enriched is broken.
- Assumption: fct_orders materialization depends directly on int_order_enriched (reasonable given they share time boundaries and the note states "fct_orders has not refreshed" since the failure).

3) Determine blast radius

- Downstream consumers listed: Looker Revenue Overview and exec_daily_orders Slack bot.
- Since fct_orders hasn't refreshed since 09:12, any consumer relying on fct_orders freshness is stale.
- Assumption: Looker and the Slack bot both consume fct_orders (or a downstream model that depends on it), so both are affected equally.

4) Prescribe ordered steps to restore within SLA

- Stabilize: confirm upstream dependencies are healthy (staging models are fine).
- Diagnose: determine root cause of int_order_enriched failure and attempt quick remediation (resource bump, retry).
- Restore: either retry int_order_enriched successfully and then trigger fct_orders, or fallback to a recent good state if retry fails.
- Validate: confirm fct_orders freshness and downstream visibility.
- Communicate: incident comms and postmortem to avoid recurrence.

5) Key assumptions called out

- Dependency graph: int_orders_enriched is a direct dependency of fct_orders (no intermediate skipping possible).
- Downstream consumers depend on fct_orders freshness (not on some earlier snapshot), so they're blocked.
- The failure mode (DBT_RUN_TIMEOUT) is resolvable by resource adjustment or retry within the time window.
- No other hidden upstream failures exist beyond what's shown in the metadata.
- We can safely rerun or backfill int_order_enriched without breaking lineage or causing data duplication issues; idempotency/retries are allowed.

6) Risk and fallback if the quick retry fails

- If int_order_enriched can't be fixed in ~30-40 minutes, fallback to the last successful int_orders dataset and run fct_orders against that snapshot, then explicitly notify consumers of slightly older data while the full fix is pursued.

This keeps the plan aligned with the 120-minute SLA: diagnose and retry first, backfill/fallback second, validate and communicate throughout.

Engineering counterpart

2:34 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

2:34 AM - Responded in 3s

Before promoting any fix to production I would verify (in staging or via a controlled canary):

1) Lineage and job definitions

- Confirm `fct_orders` directly depends on `intermediate.int_order_enriched` in the DAG and there are no hidden overrides/snapshots that could bypass it.
- Check that materialization type (incremental vs. rebuild) and merge strategies are correct and won't cause duplicates or key conflicts.

2) Data contract and compatibility

- Ensure schema of `int_order_enriched` matches what `fct_orders` expects (columns, types, not-null guarantees, keys).
- Verify partitioning/clustering and sort orders are compatible to avoid performance regression.

3) Freshness and SLAs

- Confirm the refreshed `fct_orders` will meet the 120-minute SLA end-to-end (`int_order_enriched` finish -> `fct_orders` run -> downstream visibility).
- Set/validate freshness checks and alert thresholds on `fct_orders` and `int_order_enriched`.

4) Backfill and idempotency

- Dry-run a backfill for a safe date range to ensure idempotent behavior and no double-counting.
- Confirm watermarking/incremental logic won't overwrite recent good data.

5) Resource sizing and timeouts

- Validate warehouse size, concurrency, and timeout settings are sufficient for `int_order_enriched` and `fct_orders` under load.
- Check slot availability and cost guardrails to avoid another timeout.

6) Downstream impact

- Run a quick smoke test in Looker or against the refreshed `fct_orders` to confirm metrics in "Looker Revenue Overview" and "exec_daily_orders Slack bot" are correct.
- Verify no consumer jobs will fail on schema changes.

7) Error and rollback plan

- Confirm rollback steps (e.g., revert to last successful `int_order_enriched` + `fct_orders` run) are documented and tested.
- Ensure logs and alerts surface timeout or failure causes clearly for fast triage.

8) Test coverage and monitoring

- Add or update tests for key assertions (row counts, uniqueness of keys, freshness, not-null columns).
- Put a post-deploy monitor in place for at least one full SLA cycle.

If all of the above checks pass in staging/canary, ship to production with a short communication and monitoring window.

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 3s

Top risks

- Incomplete or corrupt backfill: Re-running `int_order_enriched` could produce partial/bad data and break `fct_orders` or downstream metrics.
- Time-to-fix exceeds SLA: Diagnose/retry takes longer than the ~60-90 minutes we have, so `fct_orders` remains stale past the 120-minute SLA.
- Resource contention: Re-running `int_order_enriched` and `fct_orders` concurrently competes for warehouse slots, causing timeouts or further failures.
- Downstream breakage: Looker/Slack bot queries fail or show misleading numbers if schema or semantics change unexpectedly.
- Hidden data dependencies: Other unlisted upstreams or snapshot dependencies cause anomalies when `int_order_enriched` is re-run.
- Idempotency issues: Duplicate runs create double-counted orders or violate key constraints.

Rollback plan (fast, prioritized)

- T+0-10: Pause any in-flight backfills; open incident channel and notify stakeholders (platform, revenue owners).
- T+10-30: If the current fix attempt is mid-flight and not stabilizing, cancel it and revert warehouse/resource changes.
- T+30-60: Restore `fct_orders` and `int_order_enriched` to the last known-good state:
 - Option A (preferred): Promote the most recent successful dataset versions of `intermediate.int_order_enriched` and `fct_orders` and rerun `fct_orders` from that point.
 - Option B (fallback): If pipeline-as-code promotion isn't fast, point `fct_orders` to a static snapshot or a restored backup for immediate freshness, with a consumer-visible "data recency disclaimer."
- T+60-90: Re-run downstream consumers (Looker, Slack bot) against the restored snapshot and confirm metrics stabilize.
- T+90-120: Communicate status, document root cause, and implement safeguards (see "decisions" below).

Operational safeguards to pair with rollback

- Freeze merges on `int_order_enriched` and `fct_orders` until fix is validated.
- Add a temporary high-frequency freshness monitor (5-15 minute checks) on `fct_orders` and `int_order_enriched`.
- Pre-stage resource bumps or a dedicated backfill warehouse to avoid slot contention.
- Prepare a short comms template: "We restored freshness using [Option A/B]; dashboards reflect data as of [timestamp]; we'll provide root cause by [time]."

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.