



Vantage RuntimeAI

Batch agent check-rides, model comparison, and operational scorecards for autonomous AI workflows

8.0/10

Overall score

Run summary

Scenario: Pipeline Incident Triage

Model: inclusionai/ling-2.6-flash

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

Duration: 16s

Actual run cost: \$0.000153

Projected execution cost: \$0.1010/mo (30/biz day × 22 days)

Report date: July 13, 2026 - 11:00 PM EST

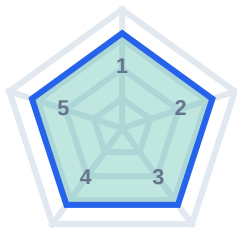
Overall

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	Root cause category	8/10
2	Uses log artifacts	8/10
3	Blast radius awareness	8/10
4	Structured triage summary	8/10
5	Next steps & verification	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.

- Root cause category identified: Met
- Restore / verification next steps: Met
- Cites log / error artifacts: 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: 6 - Counterpart msgs: 7 - Output: 10991 chars - Latency: 16s - Criteria: 3/3

Score analysis

3/3 automated criteria met - see checklist below.

Score snapshot

Solid: Solid technical task response with a few gaps.

Interpretation

Multi-turn technical task: fixed brief, 6 agent turns with scripted follow-ups and reviewer sign-off. 3 of 3 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

- + Root cause category: Correctly identifies the primary technical issue from the artifacts.
- + Uses log artifacts: Uses tables, logs, or schema facts from the task brief.
- + Blast radius awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Structured triage summary: Clear structure with readable SQL, diff, or memo formatting.
- + Next steps & verification: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Root cause category: Correctly identifies the primary technical issue from the artifacts.
- Uses log artifacts: Uses tables, logs, or schema facts from the task brief.
- Blast radius awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- Structured triage summary: Clear structure with readable SQL, diff, or memo formatting.
- Next steps & verification: 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

Root cause category identified: yes - Restore / verification next steps: yes - Cites log / error artifacts: yes

CAVEATS

- Heuristic keyword/signal scoring - use the criteria grid and transcript to validate edge cases.

Transcript

Session

Scenario: Pipeline Incident Triage

Agent response <-> Engineering counterpart

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

Duration: 16s

Transcript: blue = protagonist (you) - amber = counterpart - gray = coach/system. Each turn shows role/name, clock time, and response delay when available.

Engineering counterpart

2:59 AM

Task brief - Pipeline incident triage (complex complexity)

Triage this failed dbt run log and recommend next steps:

...

```
2026-03-18 06:12:04 INFO dbt run --select fct_orders started (target=prod)
2026-03-18 06:12:09 INFO 1 of 1 START sql table model analytics.fct_orders ..... [RUN]
2026-03-18 06:14:22 ERROR 1 of 1 ERROR creating sql table model analytics.fct_orders .. [ERROR in 133.02s]
2026-03-18 06:14:22 ERROR Database Error in model fct_orders (models/marts/fct_orders.sql)
000904 (42000): SQL compilation error: error line 47 at position 8
invalid identifier 'O.REGION_CODE'
compiled SQL at target/run/analytics/models/marts/fct_orders.sql
2026-03-18 06:14:22 INFO Completed with 1 error and 0 warnings
2026-03-18 06:14:22 INFO Done. PASS=0 WARN=0 ERROR=1 SKIP=0 TOTAL=1
```

Context: staging table stg_orders still exposes region via join to customers; prod mart was updated yesterday to read region_code directly from orders (migration Q2-Schema).

...

Agent response

2:59 AM - Responded in 2s

```
{
  "role": "assistant",
  "content": {
    "likely_root_cause_category": "Schema/object reference mismatch (upstream dependency change)",
    "what_broke_and_why": "The compiled SQL for fct_orders references O.REGION_CODE (line 47, position 8), but that column does not exist on the O table (likely renamed/removed in the Q2-Schema migration). The error is a SQL compilation error: invalid identifier 'O.REGION_CODE'.",
    "immediate_next_steps_to_restore_mart_safely": [
      "Inspect the compiled SQL at target/run/analytics/models/marts/fct_orders.sql around line 47 to confirm the invalid reference.",
      "Check the current definition of the O source/alias in the model and its dependencies (e.g., sources, staging models) to verify whether REGION_CODE exists or was replaced.",
      "If REGION_CODE was removed upstream, update fct_orders.sql to use the correct column/source (e.g., REGION or a different key) and re-run.",
      "Before re-running in prod, validate the model in a non-prod environment or dry-run to confirm compilation.",
      "If the change is part of the Q2-Schema migration, ensure all dependent models are updated and run in dependency order; then re-run fct_orders."
    ]
  }
}
```

Engineering counterpart

2:59 AM - Responded in 0s

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

Agent response

2:59 AM - Responded in 3s

Step-by-step reasoning

1) Read the error first. The dbt run failed with a SQL compilation error "invalid identifier 'O.REGION_CODE'" at line 47 position 8 in the compiled model fct_orders. That tells me the problem is in the SQL generated for this model, not infrastructure or permissions.

2) Look at context given: The prod mart was updated yesterday to read region_code directly from orders, and the staging table stg_orders still exposes region via a join to customers. This suggests someone updated the source/definition of orders in the Q2-Schema migration, likely removing or renaming the REGION_CODE column or changing how region is exposed.

3) Make minimal assumptions to form hypotheses:

- The model fct_orders references a source or intermediate node aliased as O.
- That reference expects a column REGION_CODE on O.
- Either O no longer has REGION_CODE (column removed/replaced in migration), or the alias/source definition changed so that REGION_CODE is no longer present at that position in the lineage.
- The error is a compilation error (not runtime), so the SQL was syntactically valid but references a non-existent identifier in the upstream objects.

4) From those, the likely root cause category is a schema/object reference mismatch caused by an upstream change (the Q2-Schema migration), not a network, credential, or transient infra issue.

5) What broke: the materialized model fct_orders can no longer resolve O.REGION_CODE, causing the compilation to fail and the run to error. Why: the column or its source was changed/removed in the recent migration, and dependent models were not updated accordingly.

6) Immediate next steps to restore safely:

- Examine the compiled SQL to confirm the exact statement and surrounding references.
- Inspect the model definition and its sources to verify the shape of O and whether REGION_CODE exists.
- Update fct_orders.sql (and any other dependents) to use the correct column or join path.
- Validate by compiling/running in a non-prod environment first.
- Ensure dependency order is respected across the mart and re-run the pipeline.

Assumptions I made (explicitly, to keep this focused):

- The error is SQL compilation, not runtime or infrastructure.
- The "O" alias maps to a node/source that was changed in Q2-Schema.
- REGION_CODE was removed or renamed upstream; no hidden column mappings or views exist that silently provide it.
- No other infra or network issues; the failure is reproducible from the same SQL.
- The compiled SQL referenced is available and accurate for inspection.

Engineering counterpart

2:59 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

2:59 AM - Responded in 2s

Before promoting any fix to prod I'd verify:

- Correct source/columns: Confirm the upstream object (O) actually exposes the column/alias you're referencing (REGION_CODE or its replacement) and that it exists in all expected environments.
- Cross-model impact: Check every downstream model/view that depends on fct_orders or the changed upstream to ensure they're updated and still pass.
- Compilation in isolation: Run a dbt compile of only fct_orders in prod mode (dry-run/safe) so the generated SQL is validated against the live schema without running.
- Schema parity: Compare dev/staging schema shapes (columns, types, keys) with prod to ensure no environment drift; especially confirm whether REGION_CODE exists or is named differently.
- Join/lineage correctness: Verify joins and keys between fct_orders and its sources (customers, orders, etc.) are valid and won't cause cardinality or cross-database issues.
- Tests and documentation: Ensure associated tests (not null, unique, relationships) exist and pass; update documentation/contracts if column semantics changed.
- Permissions and SLAs: Confirm the service account has required privileges and the model's materialization settings are appropriate for prod.
- Backfill/partitioning impact: If the change affects partitioning or incremental strategies, validate backfill plans and windowing to avoid data loss or heavy prod load.
- Rollback plan: Have a tested rollback (revert + re-run) and a communication checklist so ops can recover quickly if the deploy fails.

Engineering counterpart

2:59 AM - Responded in 0s

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

Agent response

3:00 AM - Responded in 3s

Top risks

- Partial or missing REGION_CODE across upstreams in prod, causing downstream join failures or null enrichments.
- Environment drift between dev/staging and prod (column name/type mismatch, different snapshot version), making the fix compile locally but fail in prod.
- Incremental strategy or partition bounds misalignment after the change, leading to missing partitions or a full-table rewrite during backfill.
- Referential integrity / relationship tests failing because keys or cardinalities changed when switching to the new source column.
- Performance regression or SLAs breach (long-running materializations, contention) when re-running large models in prod during business hours.
- Dependency ordering broken: other models that depend on fct_orders or on the changed upstream are not updated together, causing cascading errors.

Rollback plan (safe, observable, quick)

- Stop / rollback the deploy: abort any running jobs and revert the deployment artifact/model change to the prior known-good state (e.g., revert the PR/commit and redeploy the previous Docker image/tag).
- Re-run previous successful DAG:
 - Re-run the last successful run of fct_orders and any dependent models from the release pipeline, using the same snapshot/state that was previously in prod.
 - If necessary, use dbt's `dbt run-operation snapshot\_snapshot\_name` or `dbt seed` for static overrides.
- Backfill strategy (if re-running from earlier point):
 - Use dbt run with `--select` to rebuild only what's required, starting from the first changed upstream, respecting dependencies.
 - For heavy backfills, do it in non-prod first; if acceptable, run during a maintenance window or off-peak in prod with rate limits.
- Data integrity checks after rollback:
 - Re-run tests on fct_orders and its dependents (uniqueness, not-null, relationships).
 - Compare key aggregates and counts with the prior successful day-part to ensure parity.
- Operational steps:
 - Pause incoming jobs/scheduler triggers during rollback to avoid concurrency issues.
 - Capture logs and model versions; notify stakeholders and reopen a ticket with the compiled SQL and environment diff if the root cause is unclear.
- When to escalate:
 - If rollback still fails due to missing upstream columns or drift, involve data platform/infra to reconcile schema across environments before another prod attempt.

Engineering counterpart

3:00 AM - Responded in 0s

Can you tighten this into concrete next steps with owners and timing?

Agent response

3:00 AM - Responded in 4s

Concrete next steps with owners and timing (immediate, within the next 90 minutes)

- 1) Pause production jobs and freeze deploys (Owner: Release Manager / On-call SRE; Time: now)
 - Stop the scheduler (Airflow/Control-M/GitHub Actions) to prevent new runs from starting.
 - Tag/lock the current deployment so no new promotion happens until rollback completes.
- 2) Capture evidence and confirm scope (Owner: Data Engineer triage; Time: 0-15 minutes)
 - Save the compiled SQL from `target/run/analytics/models/marts/fct_orders.sql` and the failing model definition `models/marts/fct_orders.sql`.
 - Confirm the current state of the upstream source/alias O (columns and snapshot version) in prod by querying the information schema or running a quick dbt compile in dry-run mode.
 - Owner: Data Engineer; Time: 0-15 minutes.
- 3) Validate schema parity across environments (Owner: Data Engineer + Data Platform; Time: 0-30 minutes)
 - Run a schema diff between dev/staging and prod for the orders source and any customers table (columns, types, NOT NULL).
 - If mismatch found, record and plan remediation (e.g., column rename/add) with owners.
- 4) Decide and apply fix or rollback (Owner: Data Engineer lead; Time: 30-75 minutes)
 - If the fix is a simple column rename/add and the change is small:
 - Update `fct_orders.sql` (and any downstream models) to use the correct column or join.
 - Run dbt compile in prod mode (dry-run) to verify SQL.
 - Run unit/integration tests in a non-prod environment; if green, proceed.
 - If the fix is complex or risky, or schema parity fails:
 - Initiate rollback to the prior known-good image/tag and re-run the last successful DAG (see rollback plan).
- 5) Re-run in a controlled manner (Owner: Data Engineer / Scheduler Operator; Time: 75-120 minutes)
 - Re-run `fct_orders` and its direct dependencies first, then downstream consumers in dependency order.
 - Monitor run times and cancel if they threaten prod SLAs; consider running heavy backfills off-peak.
- 6) Post-run validation and sign-off (Owner: Data Engineer + Data QA; Time: 120-150 minutes)
 - Run contract tests (relationships, not-null, unique) and compare key aggregates/counts to prior successful day-part.
 - Confirm no regressions in downstream dashboards/metrics.
 - Document the root cause, change, and test results; notify stakeholders.
- 7) Resume production schedule and monitor (Owner: Release Manager; Time: ~150 minutes onward)
 - Re-enable the scheduler, resume normal triggers, and monitor for 1-2 business cycles.
 - If issues persist, escalate to Data Platform with logs and diff.

Notes on owners/roles (adjust to your team):

- Release Manager or Platform On-call: controls deploy/scheduler and rollback.
- Data Engineer (triage): owns investigation, model fix, and re-run.
- Data Platform: assists with schema parity and environment drift.
- Data QA/Analyst: owns validation and sign-off.

If you want, I can convert this into an incident runbook with exact commands (dbt, SQL checks, scheduler commands) for your stack.

Engineering counterpart

3:00 AM - Responded in 0s

What's the one thing you'd want a second pair of eyes on?

Agent response

3:00 AM - Responded in 1s

Have a second engineer confirm that the upstream object (O/orders) actually exposes the expected column and that its definition and permissions in prod match the intended contract before we re-run.

Engineering counterpart

3:00 AM - Responded in 0s

Thanks - triage ordering looks sound. Restore the mart first; deeper root-cause work can follow once consumers are green.
Review complete - no further response needed.