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: google/gemini-2.5-flash
Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 28s
Actual run cost: $0.007950
Projected execution cost: $5.25/mo (30/biz day × 22 days)
Report date: July 13, 2026 - 10:57 PM EST

Session: sample-lb-rankings-seed-cf0f2a80-a00e-4fb3-bca0-4dee15500f40

de_pipeline_incident_v1 - agent - openrouter - google/gemini-2.5-flash - ended

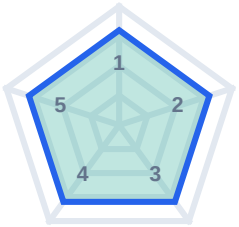
Overall

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: 17581 chars - Latency: 28s - 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: 28s

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

Engineering counterpart
2:56 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
2:57 AM - Responded in 4s

Here's the triage of the dbt production run failure:

---

1. Likely Root Cause Category:
Schema Drift / Data Model Inconsistency

2. What Broke and Why:

The dbt run failed because the `fct_orders` model is trying to access a column named `O.REGION_CODE` which does not exist in the underlying data table (or view) aliased as `O`.

Log Line Reference:
`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'`

Why it broke: The context provides the crucial piece of information: "prod mart was updated yesterday to read region_code directly from orders (migration Q2-Schema)." This indicates that the `fct_orders` model's SQL code was updated to reflect a new schema design where `REGION_CODE` is expected to be a direct column on the `orders` table (or its staging equivalent). However, the `staging table stg_orders still exposes region via join to customers`. This means the `stg_orders` model, which `fct_orders` likely depends on, has not yet been updated to include `REGION_CODE` directly from the orders source. Instead, it still expects to join to a `customers` table to get region information.

The SQL compilation error `invalid identifier 'O.REGION_CODE'` confirms that the `orders` alias (`O`) within the `fct_orders` model's underlying query does not contain a column named `REGION_CODE`, leading to the failure. The model's SQL is ahead of its dependencies' schema.

3. Immediate Next Steps to Restore the Mart Safely:

1. Revert `fct_orders` to its previous working version: The quickest way to restore the production mart is to deploy the last known good version of the `fct_orders` model. This will revert the change introduced yesterday that caused the `REGION_CODE` direct reference.
Action: Rollback the `fct_orders.sql` file in your dbt repository to the commit before the "migration Q2-Schema" update.
Action: Re-run `dbt run --select fct_orders` (or the full production run if it's part of a larger job) with the reverted code. This should allow the mart to build successfully using the old logic (joining to customers for region).

2. Investigate and Coordinate Schema Migration:
Action: Immediately investigate the state of the "migration Q2-Schema." Was the change to `stg_orders` (or the underlying source table if `stg_orders` simply selects all) supposed to happen before `fct_orders` was updated?
Action: Coordinate with the team responsible for the "migration Q2-Schema" to understand the timeline for when `REGION_CODE` will truly be available directly on the `orders` source data and subsequently reflected in `stg_orders`.
Action: Once the `stg_orders` model (or its source) is confirmed to contain `REGION_CODE` directly, then the updated `fct_orders` model can be safely deployed and run again.

In summary: The `fct_orders` model was updated prematurely, assuming a schema change (direct `REGION_CODE` on orders) that hadn'

Engineering counterpart
2:57 AM - Responded in 0s

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

Agent response
2:57 AM - Responded in 5s

Let's break down the reasoning process step-by-step, including the assumptions made.

Step-by-Step Reasoning:

1. Understand the Goal: The primary goal is to triage a failed dbt run, identify the root cause, and propose immediate next steps to restore the service (the mart) safely.

2. Initial Log Scan for Keywords:
`ERROR` is the first and most critical keyword.
`ERROR creating sql table model analytics.fct_orders`: This tells us which model failed.
`Database Error`: This indicates the error originated from the database engine itself, not a dbt parsing or configuration issue.
`SQL compilation error`: This is very specific. It means the SQL query generated by dbt was syntactically or semantically incorrect from the database's perspective. The database couldn't even start executing it.
`invalid identifier 'O.REGION_CODE'`: This is the core error message. It directly points to the problem: the database doesn't recognize a column named `REGION_CODE` when referenced via alias `O`.
`models/marts/fct_orders.sql`: Confirms the file where the error originates.

3. Synthesize Log Information:
Model `fct_orders` failed to build.
The failure was a SQL compilation error.
The specific issue was an "invalid identifier" for `O.REGION_CODE`.
This implies that in the SQL query for `fct_orders`, there's a table or view aliased as `O` which is being queried for a column named `REGION_CODE`, but that column doesn't exist in `O`.

4. Incorporate Context: This is where the provided "Context" becomes crucial.
`Context: staging table stg_orders still exposes region via join to customers;`
This tells us how `REGION_CODE` was being exposed previously: through a join to a `customers` table, likely implying `stg_orders` itself didn't have `REGION_CODE` directly.
`prod mart was updated yesterday to read region_code directly from orders (migration Q2-Schema).`
This is the smoking gun! It explicitly states that the `fct_orders` model (the "prod mart") was changed to assume `REGION_CODE` is now directly on orders.

5. Connect Log to Context (The "Aha!" Moment):
The log says `O.REGION_CODE` is an "invalid identifier" (i.e., column not found).
The context says `fct_orders` was updated to expect `REGION_CODE` directly from `orders` (which we can infer is likely aliased as `O` in the `fct_orders` SQL).
The context also says `stg_orders` (a likely upstream dependency of `fct_orders` that provides the 'orders' data) still gets region via a join to customers. This means `stg_orders` does not yet have `REGION_CODE` directly on it.

Conclusion: The `fct_orders` model was updated to use a new schema (`REGION_CODE` directly on orders), but its upstream dependency (`stg_orders` or the underlying source table it selects from) has not yet been updated to reflect that schema change. This is a classic "schema drift" scenario where the consumer (fct_

Engineering counterpart
2:57 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response
2:57 AM - Responded in 5s

Before "shipping this to production" (meaning, before deploying the recommended fix and subsequent steps), I would verify the following:

For the Immediate Fix (Reverting `fct_orders`):

1. Confirm the exact previous version of `fct_orders.sql`:
   Verification: Use Git (`git log -- models/marts/fct_orders.sql`) to identify the commit immediately preceding yesterday's "migration Q2-Schema" update. Ensure this version is indeed the last known working state.
   Rationale: We want to revert to a state that is known to work, not just any arbitrary previous version.

2. Understand the Impact of Reversion:
   Verification: Does reverting `fct_orders` have any cascading effects on other downstream models or dashboards that might have already been updated to expect data derived from the new `REGION_CODE` logic? (While unlikely for an immediate revert for a failed build, it's good practice to consider.)
   Rationale: Minimize unintended side effects. In this case, since `fct_orders` failed to build, no new data based on the problematic logic would have been generated, so this risk is low.

3. Test the Reverted Model in Staging/Dev (if time permits):
   Verification: If there's a quick way to run `dbt run --select fct_orders` in a non-production environment (e.g., a staging schema or developer environment), do so to confirm it builds successfully.
   Rationale: "Test before deploy." For a critical production incident, this might be skipped if the rollback is simple and the previous version is thoroughly trusted.

4. Communication Plan:
   Verification: Who needs to know about the rollback? The data consumers, the team working on the "Q2-Schema migration," etc.
   Rationale: Transparency and coordination are key during incidents.

For the Follow-up Investigation and Permanent Fix (Schema Migration):

1. Precise Status of `stg_orders` and Source Data:
   Verification: Directly query the tables/views that `stg_orders` selects from. Does `REGION_CODE` exist there yet? (e.g., `DESCRIBE TABLE my_source_db.orders;` or `SELECT TOP 10  FROM my_source_db.orders;`).
   Verification: Review the `stg_orders.sql` model itself. Does it currently select `REGION_CODE` directly, or does it still join to `customers`?
   Rationale: Get concrete proof of the schema state, rather than relying solely on the context statement (which might be slightly outdated or misinterpreted).

2. Migration Plan Details:
   Verification: Get the full details of the "Q2-Schema migration." What is the planned sequence of changes? Which tables are affected, and when? Who is responsible for each step?
   Rationale: Understand the overall project to ensure the dbt changes align with the data source changes.

3. Dependency Analysis:
   Verification: Are there any other models that also depend on `stg_orders` that might be impacted by the `REGION_CODE` change? Have they been updated or are they slated for update?
   Rationale: Prevent future cascades of schema drift errors.

4. Testing Strategy for the New Schema:
   Verification: Once `stg_orders` is updated to include `REGION_CODE` directly, what

Engineering counterpart
2:57 AM - Responded in 0s

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

Agent response
2:57 AM - Responded in 5s

Let's break down the top risks for both the immediate fix and the long-term solution, along with the rollback plan if the immediate fix doesn't work.

Top Risks

For the Immediate Fix (Reverting `fct_orders`):

1. Risk: Incorrect Previous Version Identified.
   Description: The most significant risk is reverting to a version of `fct_orders.sql` that wasn't the last known good state, or that had its own subtle bugs. This could lead to a different failure or incorrect data being generated.
   Impact: Continued pipeline failure, incorrect data reported to stakeholders, loss of trust.

2. Risk: Hidden Dependencies / Unexpected Side Effects.
   Description: While unlikely given the failure was a compilation error (meaning no bad data was written by this specific model), there's a small chance that other dbt models or downstream systems might have been partially affected or are now expecting an intermediate state that the revert changes.
   Impact: Breaks other data marts or dashboards, requiring further investigation and fixes.

3. Risk: Communication Breakdown.
   Description: Not properly informing stakeholders (data consumers, upstream/downstream teams) about the rollback and the temporary reinstatement of older logic.
   Impact: Confusion, incorrect business decisions based on misunderstood data, eroded trust.

For the Long-Term/Permanent Fix (Schema Migration & dbt Model Updates):

1. Risk: Misalignment of Schema Changes.
   Description: The "Q2-Schema migration" for the underlying `orders` table (or its source) doesn't perfectly align with the `stg_orders` dbt model update. For example, `REGION_CODE` might be named differently, have a different data type, or still not be directly available as expected.
   Impact: New dbt failures, incorrect data, significant rework.

2. Risk: Incomplete Migration.
   Description: The migration might only partially address the schema change (e.g., `REGION_CODE` is added, but the old method of deriving it via a `customers` join is not fully removed or causes conflicts).
   Impact: Data quality issues, performance degradation, or intermittent failures.

3. Risk: Regression in `stg_orders` or `fct_orders`.
   Description: During the process of updating `stg_orders` to reflect the new primary source, or `fct_orders` to use the new `stg_orders`, new bugs or logical errors are introduced.
   Impact: New data quality issues, incorrect metrics, or pipeline failures.

4. Risk: Data Latency/Availability During Migration.
   Description: The actual schema migration of the source table might involve downtime or a period of inconsistency for the underlying data, impacting the ability of dbt to successfully run.
   Impact: Delayed data availability, missed SLAs.

Rollback Plan if the Immediate Fix (Reverting `fct_orders`) Doesn't Work

If, for some reason, reverting `fct_orders` to its previous version and re-running the dbt model still fails (or introduces a new,

critical issue), here's the rollback plan:

1. Immediately Halt Further Deployments/Changes:
   Action: Stop any active deployment pipelines related to this dbt project to prevent introducing more variables.

2. Analyze the New Failure:
   Action: Review the logs thoroughly for the

Engineering counterpart
2:57 AM - Responded in 0s

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

Agent response
2:57 AM - Responded in 5s

Okay, let's consolidate this into concrete, actionable next steps with suggested owners and timing, focusing on both immediate restoration and the underlying problem resolution.

Incident: dbt `fct_orders` Production Failure
Severity: High (Production Mart failure)

Immediate Goal: Restore `fct_orders` mart to a working state ASAP.
Long-Term Goal: Resolve the root cause (schema drift) and implement the Q2-Schema migration correctly.

Phase 1: Immediate Restoration (Within 1-2 Hours)

Objective: Get `fct_orders` building successfully using its last known good configuration.

1. Stop Further Production Deployments for `fct_orders`
   Action: Temporarily disable/block any automated deployments or manual changes to `fct_orders` in the production environment.
   Owner: Data Engineering Lead
   Timing: Immediately

2. Identify Last Known Good Version of `fct_orders.sql`
   Action: Use Git (`git log -- models/marts/fct_orders.sql`) to pinpoint the commit before yesterday's "migration Q2-Schema" update that introduced the `REGION_CODE` direct reference. Confirm this was the last successful production run's version.
   Owner: Data Engineer On-Call
   Timing: Within 15 minutes of triage start.

3. Revert `fct_orders.sql` in Production Environment
   Action: Rollback the `fct_orders.sql` file in the production dbt repository to the identified last known good version.
   Owner: Data Engineer On-Call
   Timing: Within 30 minutes of triage start.

4. Execute Production `dbt run` for `fct_orders`
   Action: Trigger a `dbt run --select fct_orders` on the production environment with the reverted code. Monitor logs closely for success.
   Owner: Data Engineer On-Call
   Timing: Immediately after code revert and deployment.

5. Verify Mart Restoration & Communicate
   Action: Confirm `fct_orders` built successfully. Notify relevant data consumers, analytics teams, and product owners of the temporary rollback and successful restoration. Explain that `REGION_CODE` is currently sourced via `customers` join.
   Owner: Data Engineer On-Call
   Timing: Within 15 minutes of successful dbt run.

Phase 2: Root Cause Resolution & Permanent Fix (Within 1-3 Days)

Objective: Understand why the schema change was out of sync, and correctly implement the Q2-Schema migration.

1. Investigate Schema Migration Discrepancy

   Action:

   Review `stg_orders.sql`: Confirm it still sources `REGION_CODE` via a `customers` join.

   Query Source System: Directly inspect the upstream `orders` source table (e.g., `DESCRIBE TABLE source_db.orders;`) to determine if `REGION_CODE` is actually present in the source system yet.

   Consult Q2-Schema Migration Team: Meet with the team responsible for the "Q2-Schema migration" to understand its exact status, timeline, and whether `stg_orders` (or its underlying source) was supposed to be updated in coordination.

   Owner: Data Engineer (assigned after incident)

Engineering counterpart
2:57 AM - Responded in 0s
What's the one thing you'd want a second pair of eyes on?

Agent response
2:57 AM - Responded in 3s
The one thing I'd want a second pair of eyes on is:

"Confirmation of the exact last working Git commit for `fct_orders.sql` before yesterday's deployment AND verification of the content of that commit."

Why this is critical:

Foundation for the Fix: This is the absolute bedrock of the immediate rollback plan. If we revert to the wrong commit, we might either still fail, introduce a new bug, or deploy an older, irrelevant version of the model.
Preventing Further Damage: Incorrectly identifying the rollback point could lead to continued production failures or data inconsistencies, exacerbating the incident.
Context for Root Cause: Understanding what exactly was in the working version compared to the failing version is crucial for understanding the precise nature of the schema drift and the expected state.
Simple yet Prone to Error: While seemingly straightforward, Git history can sometimes be complex (merges, reverts, squashes). A fresh pair of eyes double-checking the `git log` output, the commit diff, and potentially even a quick comparison of the file's contents between the two versions minimizes human error under pressure.

Having a second person independently confirm "Yes, this specific commit hash is the one we should revert to, and I've quickly reviewed its contents to ensure it looks like the expected working version" would significantly increase confidence in the immediate fix.

Engineering counterpart
2:57 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.