



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/gemma-3-12b-it

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

Duration: 1m 26s

Actual run cost: \$0.000765

Projected execution cost: \$0.5049/mo (30/biz day × 22 days)

Report date: July 13, 2026 - 10:58 PM EST

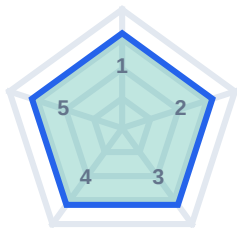
Overall

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: 18497 chars - Latency: 86s - 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: 1m 26s

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

Engineering counterpart

2:57 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 16s

Okay, here's my triage of the dbt production run failure, broken down into the requested categories.

(1) Likely Root Cause Category: Data Model/Schema Change & Code Incompatibility

(2) What Broke and Why (Referencing Log Lines):

What Broke: The ``analytics.fct_orders`` model failed to compile and run in production. This is indicated by the ``ERROR creating sql table model analytics.fct_orders`` and ``Completed with 1 error`` log lines.

Why: The error stems from a SQL compilation error due to an "invalid identifier 'O.REGION_CODE'". This directly indicates a problem with the SQL code within the ``models/marts/fct_orders.sql`` file.

Contextual Explanation: The provided context is critical. It states, "prod mart was updated yesterday to read `region_code` directly from orders (migration Q2-Schema)." This reveals the core issue: a schema change impacted the code. The code now expects ``REGION_CODE`` to be directly available in the ``orders`` table (aliased as ``O`` in the query). However, the staging table ``stg_orders`` still exposes region through a join to the ``customers`` table. The new ``fct_orders.sql`` code likely removed the join to ``customers`` and assumed ``REGION_CODE`` was a direct column on ``orders``. The error "invalid identifier" means the database engine cannot find the ``REGION_CODE`` column in the context of the query.

(3) Immediate Next Steps to Restore the Mart Safely:

Given the situation, the safest and quickest path to restoration is a rollback. Here's the plan:

1. Rollback the ``fct_orders.sql`` Model: Revert the ``models/marts/fct_orders.sql`` file to the previous version. This version presumably worked correctly before the schema change. This can be done via your version control system (Git) if you have proper versioning in place, which you absolutely should. If not immediately available, quick manual code replacement from a known good source (like a development or testing environment) is necessary.
2. Run dbt ``run --select fct_orders`` again: After reverting the model, trigger a dbt run specifically targeting ``fct_orders`` in the production environment. This isolates the fix and ensures the mart is rebuilt correctly using the older SQL.
3. Verify the Data: After the run completes successfully, immediately validate the data in the ``analytics.fct_orders`` mart. Compare a few rows with data from before yesterday's change to ensure the rollback restored the expected values. Automated data quality checks are ideal for this, but manual spot checks are acceptable in the immediate aftermath of an incident.
4. Investigate the Root Cause (Post-Restoration): After the mart is restored and verified, thoroughly investigate the root cause.

Review the Schema Migration: Understand why the change was made. What was the goal of migrating ``region_code`` directly from the ``orders`` table?

Code Review: Examine the updated ``fct_orders.sql`` code. Did it adequately handle the potential for the ``region_code`` column not being available in the orders table if the migration wasn't fully completed?

Staging Environment Validation: Ensure that the staging environment accurately reflects the production environment before deploying changes. The staging environment should have already included the migration from ``customers`` to ``orders``.

Versioning and Testing: Strengthen your d

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 17s

Okay, let's break down my reasoning step-by-step, along with the assumptions I made, for this dbt production incident triage.

Phase 1: Initial Assessment - Understanding the Log

1. Initial Observation: The first thing I did was parse the log lines. I saw a failure during the `dbt run` process, specifically targeting the `analytics.fct\_orders` model. This immediately flags a problem with that model.
2. Error Message Focus: The most crucial part was the `Database Error in model fct\_orders...SQL compilation error: error line 47 at position 8 invalid identifier 'O.REGION\_CODE'`. This precise error message tells me:

The problem is a compilation error - the SQL code itself is flawed, not a runtime data issue.

It's an "invalid identifier." The database cannot find a column named `REGION\_CODE` in the context where it's being used (aliased as 'O', presumably the `orders` table).

3. Severity Assessment: Because it's a compilation error, the entire model failed. This isn't a partial failure; the mart isn't being populated at all.

Phase 2: Contextualizing with Provided Information - The Key Insight

1. The Context is Huge: The provided context, "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)," is the heart of solving this. Without this, I'd be guessing wildly.
2. Connecting the Dots: This context directly explains the "invalid identifier" error. It states that yesterday, the code was changed to assume `region\_code` existed directly on the `orders` table. The fact that the staging table still requires a join to `customers` reveals a disconnect. The production environment hasn't fully caught up with the planned schema migration.
3. Forming a Hypothesis: My hypothesis became: "The updated `fct\_orders.sql` code removed the join to the `customers` table, expecting `REGION\_CODE` to be available directly from `orders`. Production hasn't been updated to reflect that change."

Phase 3: Root Cause Categorization

1. Eliminating Other Possibilities: The error points squarely to a SQL code issue. Infrastructure problems (database connectivity, resource constraints, etc.) weren't indicated-the error message is specific to the SQL itself.
2. Categorization: Based on this, the most accurate category is Data Model/Schema Change & Code Incompatibility. It's not a data quality issue (bad data causing an error); it's a mismatch between the code and the underlying data structure.

Phase 4: Recommended Next Steps - Prioritizing Stability and Risk Mitigation

1. Prioritization: The primary goal is to restore service as quickly as possible. A broken mart prevents downstream users from accessing critical data.
2. Rollback as the Safest Option: Given the recent change, a rollback to the previous, known-working version of the model is the safest and fastest route. This avoids introducing further complexity and potential errors. Trying to debug the SQL live in production is high-risk.
3. Targeted Run: Running `dbt run --select fct\_orders` isolates the problem and limits the impact of the fix. I'm not rebuilding everything; just the broken mart.
4. Verification is Critical: Data validation immediately after the restoration is non-negotiable. Even

Engineering counterpart

2:57 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

2:58 AM - Responded in 16s

Okay, you're right to push for a more thorough verification before shipping the rollback to production. Just restoring the mart isn't enough - we need to be confident the solution is stable and won't cause new issues. Here's a breakdown of what I'd verify before declaring the incident resolved and shipping the fix to production, categorized by level of effort/impact:

1. Essential & Quick Verification (Must Do - ~15-30 mins)

Build Confirmation: After rolling back the code and running ``dbt run --select fct_orders``, absolutely confirm that the model built successfully without any errors. Double-check the dbt logs to be 100% sure. This is a sanity check.

Basic Data Sampling (Row Count / Key Metrics):

Compare the row count of ``analytics.fct_orders`` now with a snapshot taken before yesterday's change. They should be virtually identical (accounting for any genuinely new data in the interim). A significant discrepancy indicates something more went wrong.

Check a few key metrics within the mart. For example, if ``fct_orders`` tracks total order value, check the total order value now vs. before. This helps catch subtle data distortions.

Visual Inspection of a Few Records: Manually inspect 5-10 records in the mart. Look at the ``REGION_CODE`` column (and other key columns) to ensure they have plausible values and align with your expectations.

2. Moderate Effort Verification (Highly Recommended - ~30-60 mins)

Data Quality Checks (Existing): Do you have existing data quality checks (dbt tests, Great Expectations, etc.) defined for ``analytics.fct_orders``? Run them. These tests can automatically validate data integrity. If you don't have any, this incident is a strong motivator to add them.

Downstream Dependency Validation: Identify any critical downstream models or dashboards that directly depend on ``analytics.fct_orders``. Briefly check those dependencies to see if they now refresh successfully and display data correctly. Focus on dependencies used by high-impact stakeholders.

Staging Environment Comparison: Rebuild ``analytics.fct_orders`` in the staging environment using the rolled-back code. Then, compare the data in ``analytics.fct_orders`` in staging with production (from before the incident). This helps confirm the code and data are consistent between environments. This is a great way to reveal if there are lingering schema differences.

3. More Extensive Verification (If Time Allows & High Risk - ~1-2 hours+)

Full Regression Testing: If you have a comprehensive regression test suite for your dbt project, run it. This will validate that the rollback hasn't inadvertently broken any other models.

User Acceptance Testing (UAT) - Limited Scope: If possible, engage a small group of key users to briefly review the data in the mart and confirm it meets their needs. This is especially important if there are complex business rules encoded in the mart.

Monitoring Setup: Set up temporary increased monitoring on ``analytics.fct_orders`` and its dependencies. Monitor for unexpected query performance changes or data anomalies. This will help catch any lingering issues that weren't caught in earlier testing.

Documentation Before Shipping:

Incident Summary: Document the incident details, root cause, rollback steps taken, and verification results.

Communication Plan: Prepare a communication plan to inform stakeholders about the incident and resolution.

Important Considerations:

Risk Tolerance: The level of verification you perform should be proportional to the risk associated with the

Engineering counterpart
2:58 AM - Responded in 0s

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

Agent response

2:58 AM - Responded in 16s

Okay, let's break down the top risks associated with this rollback and then outline a rollback plan for the rollback itself (a "rollback of the rollback," essentially!).

Top Risks Associated with the Initial Rollback

1. **Data Inconsistency (Subtle Data Corruption):** The most concerning risk is that the rollback appears to work, but introduces subtle data inconsistencies that aren't immediately obvious. This could be due to:

Partial Data Updates: If the data pipeline has been running any amount of time since the original change, there's a chance some data has been partially updated based on the new schema. Rolling back now might leave a mix of old and new data, leading to inconsistencies.

Accumulated Incremental Changes: Even small incremental updates to source systems could lead to discrepancies. The rolled-back model might not correctly process these changes.

2. **Downstream Dependency Breakage (Cascading Failures):** While we're checking critical downstream dependencies, there's a risk that other, less-obvious dependencies are broken by the rollback. This could create a cascade of failures across your data ecosystem.

3. **Missed Edge Cases in Data Validation:** Our initial data sampling and tests might not catch all edge cases or complex business rules that are affected by the schema change.

4. **Performance Degradation:** The rolled-back model might be less efficient than the updated model, leading to slower query performance. This might not be immediately apparent, but could become an issue under load.

5. **Code Conflicts & Versioning Issues:** Although we're assuming a clean rollback, there's always a small risk of code conflicts or versioning issues that prevent the rollback from completing successfully.

Rollback Plan for the Rollback (Plan B)

Let's assume the initial rollback doesn't completely resolve the issue - say, we see data inconsistencies, downstream dependencies fail, or performance degrades significantly. Here's our Plan B:

1. **Pause & Isolate:** Immediately pause further data pipeline processing. We don't want to make the situation worse by accumulating more potentially inconsistent data.

2. **Revert to Previous State:** Manually revert the ``models/marts/fct_orders.sql`` file to its original broken state (the state that caused the initial error). This ensures we're back to where we started, eliminating any lingering partial updates.

3. **Diagnosis & Root Cause Deep Dive:**

Detailed Data Comparison: Perform a more thorough data comparison between the broken mart and the state before the initial change. Use SQL queries to identify specific discrepancies.

Source System Investigation: Investigate the source systems and data flows involved. Is the schema migration in the source systems complete? Are there any data quality issues in the source systems that are masking the actual problem?

4. **Temporary Fix (If Possible & Urgent):**

Code Patch (Temporary): If the root cause is relatively simple (e.g., a missing column name), it might be possible to apply a small, targeted code patch to the broken model to temporarily work around the issue. Important: This is a temporary fix only - it shouldn't be considered a permanent solution. It should be documented clearly as such.

5. **Long-Term Solution & Full Migration:** The ultimate solution is to complete the schema migration in both the source systems and the dbt model. This involves:

Updating the ``stg_orders`` model to reflect the new schema.

Updating the ``fct_orders`` model to correctly consume the new schema.

Thorough testing and validation.

Contingency Resources & Communication

Engineering counterpart
2:58 AM - Responded in 0s

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

Agent response

2:58 AM - Responded in 14s

Okay, let's solidify this into concrete next steps with owners and timing. This assumes a team of roughly 3-5 data engineers. Please note: These timings are estimates and depend heavily on your team's expertise, tooling, and complexity of the data. This plan focuses on immediate stabilization.

Incident: `analytics.fct\_orders` Production Failure - Rollback & Stabilization Plan

Overall Goal: Restore `analytics.fct\_orders` mart to a stable state and prevent further data inconsistencies.

Phase 1: Immediate Rollback & Basic Verification (Target: 30-60 mins)

Step	Description	Owner(s)	Timing (Est.)	Dependencies
1.1	Revert `models/marts/fct_orders.sql` to previous version (via Git)	Data Engineer 1	5 mins	Git repository access
1.2	Run `dbt run --select fct_orders --progress`	Data Engineer 1	10 mins	dbt CLI, access to production environment
1.3	Verify build success in dbt logs.	Data Engineer 1	2 mins	dbt logs access
1.4	Check `analytics.fct_orders` row count vs. pre-incident snapshot (if available)	Data Engineer 2	5 mins	Access to database, Snapshot data
1.5	Spot check 5-10 records in `analytics.fct_orders` for plausibility.	Data Engineer 2	8 mins	Database access, Business context understanding

Phase 2: Moderate Verification & Dependency Check (Target: 1-2 hours)

Step	Description	Owner(s)	Timing (Est.)	Dependencies
2.1	Run existing dbt tests for `analytics.fct_orders` (if any).	Data Engineer 3	15 mins	dbt tests defined, dbt CLI
2.2	Identify 2-3 critical downstream dashboards/models that rely on `analytics.fct_orders`.	Data Engineer 3, Analyst 1	10 mins	Data lineage documentation
2.3	Briefly check downstream dashboards for successful refresh and data validity.	Analyst 1	20 mins	Dashboard access, Business context
2.4	Rebuild `analytics.fct_orders` in staging environment with rolled-back code.	Data Engineer 1	15 mins	Staging environment access
2.5	Compare data between staging `analytics.fct_orders` and production snapshot (pre-incident).	Data Engineer 2	20 mins	Database access, Snapshot data

Phase 3: Contingency - Rollback of Rollback (If Phase 2 Reveals Issues - Initiate Immediately)

Step	Description	Owner(s)	Timing (Est.)	Dependencies
3.1	Pause all data pipeline processing related to `analytics.fct_orders`.	Data Engineer 1, DevOps	5 mins	Pipeline control mechanisms
3.2	Revert `models/marts/fct_orders.sql` back to the original broken state (Git).	Data Engineer 1	5 mins	Git repository access

Engineering counterpart
2:58 AM - Responded in 0s

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

Agent response

2:58 AM - Responded in 6s

The single most critical thing I'd want a second pair of eyes on is Step 2.5: Compare data between staging `analytics.fct\_orders` and production snapshot (pre-incident).

Here's why:

Subtle Inconsistencies are the Biggest Risk: As I outlined earlier, the biggest risk is subtle data corruption that isn't immediately obvious. This is extremely difficult to catch with just row counts or basic spot checks.

It's Diagnostic: This comparison isn't just about verifying the result of the rollback; it's about digging into why things might be going wrong. Discrepancies here will point directly to the root of the problem - whether it's partial updates, incremental changes, or lingering schema differences.

Requires Domain Expertise: A second engineer brings a fresh perspective and potentially a different understanding of the business logic and data relationships within the mart. They might spot patterns or anomalies that the original engineer missed.

Reduces Cognitive Bias: It helps avoid confirmation bias. The original engineer might be focused on confirming their understanding of the rollback, while a second pair of eyes brings a more objective viewpoint.

Ideally, I'd want a data engineer with strong SQL skills and a good understanding of the business context of `analytics.fct\_orders` to review this step. Their expertise would be invaluable in identifying and interpreting any discrepancies found.

Engineering counterpart

2:58 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.