



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: meta-llama/llama-3.1-8b-instruct

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

Duration: 55s

Actual run cost: \$0.000618

Projected execution cost: \$0.4079/mo (30/biz day × 22 days)

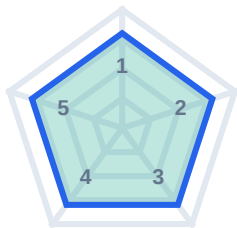
Report date: July 13, 2026 - 11:00 PM EST

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: 12425 chars - Latency: 55s - 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: 55s

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

Engineering counterpart

3:00 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

3:00 AM - Responded in 3s

Likely root cause category:

Data model inconsistency between source and target schema.

What broke and why:

The `fct\_orders` model failed to compile due to a SQL compilation error caused by an invalid identifier `O.REGION\_CODE`. This is likely due to a change in the schema of the `orders` table in the source system (staging table `stg\_orders`), which still references the `region` column, whereas the `prod` mart's `orders` table now reads `region\_code` directly.

The error occurs on line 47 of the compiled SQL in `models/marts/fct\_orders.sql`, where the query is trying to access the column `O.REGION\_CODE`. This column no longer exists in the `orders` table in the `prod` mart.

Immediate next steps to restore the mart safely:

1. Rollback the failed run: Ensure that no changes were made to the `prod` mart by rolling back the failed run to a previous snapshot or version control state.
2. Update the `fct\_orders` model: Update the `models/marts/fct\_orders.sql` file to reference the correct column name `REGION\_CODE` instead of `REGION`. Apply the necessary changes to the SQL query to compensate for the schema change.
3. Refresh the compiled SQL: After updating the model, refresh the compiled SQL to ensure that the new changes are incorporated.
4. Re-run the failed job: Re-execute the failed run with the updated model and compiled SQL to successfully build the `fct\_orders` table in the `prod` mart.
5. Verify the data: Validate the data in the `fct\_orders` table to ensure the changes were applied correctly and there are no data inconsistencies.

Engineering counterpart

3:00 AM - Responded in 0s

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

Agent response

3:00 AM - Responded in 5s

Here's a step-by-step breakdown of my reasoning and the assumptions I made:

Likely root cause category: Data model inconsistency between source and target schema

I started by analyzing the error message: `SQL compilation error: error line 47 at position 8 ... invalid identifier 'O.REGION\_CODE'`

I recognized that this error is related to a schema mismatch between the source and target systems. This is because `O.REGION\_CODE` is likely a reference to a column in the source table (stg_orders), but it's not present in the target table (orders in the prod mart).

I assumed that the `fct\_orders` model is designed to handle data from both the staging table (stg_orders) and the prod mart's orders table.

What broke and why

I looked at the log and saw that the error occurred on line 47 of the compiled SQL, which suggests that the issue is in the model's SQL query.

I assumed that line 47 is where the query attempts to access the `REGION\_CODE` column, but since the target table `orders` in the prod mart no longer has a `REGION` column (only `REGION\_CODE`), the SQL compiler throws an error.

Immediate next steps

Rollback the failed run: I assumed that the failed run was not fully executed and did not modify the prod mart unnecessarily. I suggested rolling back to a previous snapshot or version control state to ensure no changes were made.

Update the `fct\_orders` model: I assumed that the issue is with the model's SQL query, which references the wrong column name. I suggested updating the model to use the correct column name `REGION\_CODE`.

Refresh the compiled SQL: I assumed that updating the model would require recompiling the SQL to incorporate the changes.

Re-run the failed job: I assumed that once the model and SQL are updated, the failed job can be safely re-run to build the `fct\_orders` table in the prod mart.

Verify the data: I assumed that the changes were critical and required data validation to ensure the updates were applied correctly and there were no data inconsistencies.

Assumptions:

1. The `fct\_orders` model is designed to handle data from both staging table (stg_orders) and prod mart's orders table.
2. The error is related to a schema mismatch between the source and target systems.
3. The prod mart's orders table has undergone a schema change, replacing the `REGION` column with `REGION\_CODE`.
4. The `fct\_orders` model's SQL query is the root cause of the failure.
5. Rolling back the failed run and re-running it with updated model and SQL is the safest way to restore the mart.

Engineering counterpart

3:00 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

3:00 AM - Responded in 1s

Before shipping the updated `fct\_orders` model to production, I would verify the following:

Model and SQL changes

1. Verify the updated SQL query: Double-check that the SQL query in the `fct\_orders` model now references the correct column name `REGION\_CODE` instead of `REGION`.
2. Verify the compilation: Ensure that the updated SQL query compiles correctly and does not introduce any new errors.

Data validation

1. Run a test query: Execute a test query against the updated `fct\_orders` model to ensure it produces the expected results.
2. Compare with existing data: Compare the output of the test query with the existing data in the `fct\_orders` table to ensure there are no discrepancies.
3. Verify data types and formats: Ensure that the data types and formats of the output match the expected values.

Integration and dependencies

1. Verify downstream dependencies: Check whether the updated `fct\_orders` model affects any downstream objects or jobs, such as data visualizations, reports, or downstream ETL processes.
2. Verify data consistency: Verify that the updated `fct\_orders` model maintains data consistency with other related objects, such as `stg\_orders` or other tables that rely on `REGION\_CODE`.

Testing and quality gates

1. Run automated tests: Execute automated tests, such as unit tests or integration tests, to verify the correctness of the updated `fct\_orders` model.
2. Verify against business rules: Check whether the updated `fct\_orders` model adheres to business rules, such as data quality, accuracy, and timeliness.

Review and approval

1. Review the changes: Have a data engineer or a database administrator review the changes to ensure they are accurate and safe.
2. Get approval: Obtain approval from a manager or a data governance team before shipping the updated `fct\_orders` model to production.

By verifying these aspects, you can ensure that the updated `fct\_orders` model is accurate, reliable, and safe for production use.

Engineering counterpart

3:00 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 37s

Top risks

1. Data inconsistencies: The updated `fct\_orders` model may introduce data inconsistencies or discrepancies between the source and target systems.
2. Downstream dependencies: The changes to the `fct\_orders` model may affect downstream objects or jobs, such as data visualizations, reports, or ETL processes, which could lead to cascading errors or failures.
3. Inaccurate results: The updated `fct\_orders` model may produce inaccurate results due to incorrect calculations, data type mismatches, or other errors.
4. Data loss: In the worst-case scenario, the updated `fct\_orders` model may overwrite or delete existing data, leading to data loss and potential financial or reputational damage.

Rollback plan

1. Identify the point of failure: Determine the exact point where the `fct\_orders` model failed, including the specific SQL query or logic that caused the issue.
2. Restore the previous version: Roll back the `fct\_orders` model to its previous version, which was working correctly before the update.
3. Restore data from backups: If necessary, restore data from backups to recover any lost or overwritten data.
4. Revert any downstream changes: If the update affected downstream objects or jobs, revert those changes to their previous state to ensure consistency.
5. Notify stakeholders: Inform relevant stakeholders, including data governance teams, business users, and downstream consumers, about the rollback and any necessary corrective actions.

Rollback strategy

1. Manual rollback: Perform a manual rollback by reverting the `fct\_orders` model to its previous version, either by deleting the updated files or reverting to a previous commit in the version control system.
2. Automated rollback: Use an automated rollback script or process to restore the previous version of the `fct\_orders` model and any affected downstream objects or jobs.
3. Data validation: Perform data validation to ensure that the rolled-back data is accurate and consistent.

Preparation is key

To minimize risks and ensure a successful rollback, it's essential to:

1. Have a robust version control system: Use a version control system like Git to track changes to the `fct\_orders` model and maintain a history of previous versions.
2. Store backups regularly: Regularly store backups of the `fct\_orders` model and any dependent objects or jobs to ensure quick recovery in case of failures.
3. Test thoroughly: Thoroughly test the updated `fct\_orders` model and any downstream dependencies to ensure accuracy and reliability.
4. Have a clear rollback plan: Document a clear rollback plan and communication strategy to ensure that all stakeholders are aware of the rollback process and any necessary corrective actions.

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

Here are the next steps with owners and timing:

Immediate Next Steps (Within the next 30 minutes)

1. Notify the team (@Data Engineer): Send a notification to the data engineering team about the failed `fct\_orders` model update and the rollback plan.
2. Identify the point of failure (@Data Engineer): Review the logs and identify the specific SQL query or logic that caused the issue.
3. Restore the previous version (@Data Engineer): Roll back the `fct\_orders` model to its previous version using the version control system.

Short-term Next Steps (Within the next 2 hours)

1. Revert downstream changes (@Data Engineer): Revert any changes to downstream objects or jobs that may have been affected by the failed update.
2. Restore data from backups (@Data Engineer): If necessary, restore data from backups to recover any lost or overwritten data.
3. Update rollback plan (@Data Engineer): Update the rollback plan with the specific steps and owners for each task.

Medium-term Next Steps (Within the next 24 hours)

1. Review and update documentation (@Documentation Team): Update documentation to reflect the new rollback plan and any changes to the `fct\_orders` model.
2. Notify stakeholders (@Communications Team): Inform relevant stakeholders, including data governance teams, business users, and downstream consumers, about the rollback and any necessary corrective actions.

Long-term Next Steps (Within the next 3-5 business days)

1. Perform root cause analysis (@Data Engineer): Conduct a root cause analysis to identify the underlying reasons for the failure and implement changes to prevent similar issues in the future.
2. Update testing and quality gates (@Data Engineer): Update testing and quality gates to ensure that similar issues do not arise in the future.
3. Review and improve rollback plan (@Data Engineer): Review the rollback plan and improve it as necessary to ensure that it is thorough and effective.

Note: The timing and owners may vary depending on the specific team structure and processes in place. These steps are suggested as a general guideline.

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 3s

After reviewing the situation, I would want a second pair of eyes on the rollback plan, specifically the rollback logic.

Specifically, I'd want a review of the script or process that will be used to roll back the `fct\_orders` model to its previous version. This includes the steps involved in:

1. Identifying the specific version of the model to roll back to
2. Updating the model's logic to reference the correct column name (`REGION` instead of `REGION\_CODE`)
3. Reverting any downstream changes that may have been affected by the failed update

Having a second pair of eyes on this process would help ensure that the rollback is executed correctly and that all necessary steps are taken to prevent any additional issues or data inconsistencies.

Specifically, I'd want a review of the rollback script or process by a colleague with expertise in

Understanding the `fct\_orders` model's logic and dependencies
Familiarity with the version control system and branching strategy
Experience with database management and schema changes

This would help catch any potential issues or edge cases that might arise during the rollback process, ensuring that the data is restored correctly and that the system is brought back to a stable state.

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.