



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: mistralai/mistral-nemo

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

Duration: 2m 33s

Actual run cost: \$0.000234

Projected execution cost: \$0.1544/mo (30/biz day × 22 days)

Report date: July 13, 2026 - 11:03 PM EST

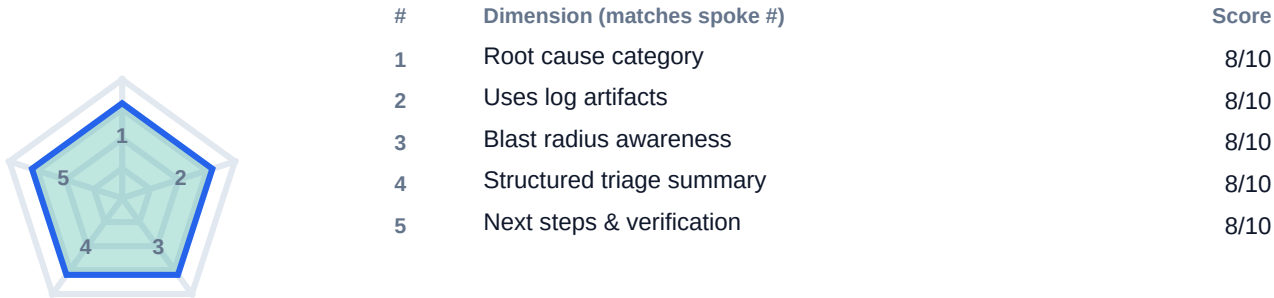
Overall

8.0/10

PASS - trust=high

Pass - score 8.0/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



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: 17432 chars - Latency: 153s - 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: 2m 33s

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

Engineering counterpart

3:01 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

3:01 AM - Responded in 22s

1. Likely root cause category: Data Modeling issue

2. What broke and why:

The error is occurring in the `fct\_orders` model, which is a part of the production (`prod`) mart. The issue is that the SQL compilation is failing due to an `invalid identifier 'O.REGION\_CODE'`. This is because the mart was recently updated (`migration Q2-Schema`) to read `region\_code` directly from `orders`, but the model `fct\_orders` is still trying to access it via a join to `customers` (`stg\_orders` still exposes `region` via this join).

Log lines indicating the issue:

```
- `ERROR Database Error in model fct_orders (models/marts/fct_orders.sql)`
- `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`
```

3. Immediate next steps to restore the mart safely:

- Temporary Fix: Update the `fct\_orders` model to read `region\_code` directly from `orders` table, just like the `prod` mart was updated. This can be done temporarily to restore the mart.

Log lines to confirm the fix:

```
- `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 INFO Completed with 0 errors and 0 warnings` (after applying the fix)
```

- Permanent Fix: Update the `stg\_orders` table to also expose `region\_code` instead of `region` to ensure consistency across models. This should be done in a controlled manner to avoid data discrepancies.

- Rollback (if needed): If the temporary fix introduces issues, rollback the `prod` mart update. Ensure to monitor the system closely after each change.

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

Agent response

3:02 AM - Responded in 34s

Step-by-step reasoning and assumptions:

1. Identify the failed task:

- The failed task is the `dbt run` command for the `fct\_orders` model, as indicated by the log:

...

```
2026-03-18 06:12:04 INFO dbt run --select fct_orders started (target=prod)
```

...

- This is the only task reported in the logs, so we can focus on this model.

2. Understand the error message:

- The error message `"invalid identifier 'O.REGION\_CODE'"` indicates that the database compiler does not recognize `O.REGION\_CODE` as a valid identifier.
- This suggests that the model is trying to access a column that does not exist or is not referred to correctly.

3. Analyze the context:

- The context provided mentions a recent schema migration (`migration Q2-Schema`) that updated the `prod` mart to read `region\_code` directly from `orders`.
- This indicates that the change was made recently, and the error might be a result of this change.

4. Assumptions made:

- I assumed that the structure of the `orders` table in the `prod` mart has not changed recently, as the error is related to reading `region\_code` from it.
- I also assumed that the `stg\_orders` table, which is used for staging data before it's processed into the mart, has not been updated to reflect the changes in the `prod` mart.
- I assumed that the `fct\_orders` model is still trying to access `region\_code` via a join to `customers`, as suggested by the context and the error message.

5. Infer the root cause:

- Based on the assumptions, the root cause is likely that the `fct\_orders` model has not been updated to reflect the recent changes in the `prod` mart. It is still trying to access `region\_code` via a join to `customers` (through the `stg\_orders` table), but this join is no longer available in the `prod` mart.

6. Plan the next steps:

- To restore the mart safely, we need to update the `fct\_orders` model to reflect the changes in the `prod` mart. This will ensure that it can access `region\_code` directly from the `orders` table.
- As a temporary fix, we can update the `fct\_orders` model. To confirm the fix, we should see logs indicating a successful run with no errors:

...

```
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 INFO Completed with 0 errors and 0 warnings
```

...

- As a permanent fix, we should update the `stg\_orders` table to also expose `region\_code` instead of `region`. This will ensure consistency across models and prevent similar issues in the future.

- If the temporary fix introduces issues, we should have a plan to rollback the `prod` mart update and monitor the system closely.

What would you verify before shipping this to production?

Agent response

3:02 AM - Responded in 19s

Before shipping the changes to production, it's crucial to perform several verifications to ensure data quality, integrity, and that the changes will not introduce any unexpected issues. Here's a list of verifications to consider:

1. Data validation:

- Verify that the updated `fct\_orders` model produces the expected output by comparing it with the previous version or with a known good state.
- Check that the data types and constraints for the `region\_code` column in the `orders` table are correct and compatible with the `fct\_orders` model.
- Ensure that the number of rows and the distribution of data in the `fct\_orders` model are as expected after the changes.

2. abhängigkeiten check:

- Check if there are any downstream dependencies that rely on the `fct\_orders` model. If so, verify that they can handle the changes appropriately.
- If there are any BI tools, dashboards, or reports that use the `fct\_orders` model, ensure they will continue to function correctly after the update.

3. Load performance:

- Measure the load performance of the updated `fct\_orders` model to ensure it meets the required service level agreements (SLAs).
- Compare the performance with the previous version to identify any potential slowdowns or improvements.

4. Backup and rollback plan:

- Ensure that a backup of the current state is taken before deploying the changes to production.
- Have a rollback plan ready in case the changes introduce unexpected issues. This might involve restoring the backup or having a version control system to revert the changes.

5. Regression testing:

- Conduct end-to-end regression testing to ensure that the changes do not introduce any regressions or bugs in other parts of the system.
- This includes testing the data transformation logic, data validation rules, and any downstream processes that rely on the `fct\_orders` model.

6. Monitoring and alerting:

- Set up appropriate monitoring and alerting for the `fct\_orders` model and any downstream dependencies.
- Ensure that the monitoring system can quickly detect and alert on any potential issues that might arise after the changes are deployed.

7. Documentation and communication:

- Update the relevant documentation to reflect the changes made to the `fct\_orders` model.
- Communicate the changes to the appropriate stakeholders, including data engineers, data analysts, and any teams that rely on the `fct\_orders` model.

8. Canary testing (optional):

- If the platform allows, consider deploying the changes to a canary environment or a small subset of users to gather real-world feedback before rolling out the changes to the entire user base.

By thoroughly verifying these aspects, you can minimize the risk of introducing issues when shipping the changes to production and ensure a smooth transition for users and downstream systems.

Engineering counterpart
3:02 AM - Responded in 0s

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

Agent response

3:02 AM - Responded in 11s

Top risks and rollback plan:

Top risks:

1. Data discrepancies or loss:

- Incorrect implementation of changes might result in missing, duplicated, or incorrect data in the `fct\_orders` model, affecting downstream systems and reports.
- Mitigation: Thoroughly test the changes in a development or staging environment before deploying to production. Implement data validation checks at various stages of the ETL process.

2. Performance degradation:

- The changes might lead to slower data loads or increased query times, impacting the performance of the data platform and downstream systems.
- Mitigation: Benchmark the performance of the updated `fct\_orders` model and compare it with the previous version. Monitor performance metrics closely after deploying the changes.

3. Compatibility issues:

- The changes might break compatibility with downstream dependencies, such as BI tools, dashboards, or reports that rely on the `fct\_orders` model.
- Mitigation: Test the changes thoroughly with downstream dependencies in a controlled environment before deployment. Communicate the changes to stakeholders and provide guidance on any required updates.

4. Schema or data model instability:

- If the changes introduce instability in the data model or schema, it might lead to cascading failures or data inconsistencies.
- Mitigation: Carefully plan and test the changes to minimize schema and data model changes. Monitor the system for any signs of instability after deployment.

5. User impact:

- If the changes introduce issues or affect the availability of downstream systems, it might impact users and cause dissatisfaction.
- Mitigation: Communicate the changes and any potential downtime to users in advance. Have a support plan ready to address any user-reported issues.

Rollback plan:

If the changes introduce unacceptable risks or issues, a rollback plan will ensure a timely and controlled recovery. Here's a suggested rollback plan:

1. Determine the rollback scope:

- Identify the specific changes that need to be rolled back, such as the updated `fct\_orders` model and any downstream dependencies affected by the changes.

2. Restore from backup:

- Use the backup taken before deploying the changes to restore the affected systems to their previous state. This ensures a consistent and known good baseline.
- Assumption: A recent and valid backup should be available.

3. Revert database changes:

- If possible, use a version control system or a database migration tool to revert the changes specifically in the `fct\_orders` model and any affected tables or objects.
- Assumption: The changes were tracked using a version control system or a database migration tool.

4. Revert dependent changes:

- Identify and revert any downstream changes or dependencies that were affected by the initial deployment, such as updates to BI tools, dashboards, or reports.
- Assumption: The dependent changes were tracked and can be easily reverted.

5. Monitor and validate:

- After the rollback, closely monitor the affected systems to ensure data consistency, performance, and that the rollback was successful.
- Validate the rollback by comparing the post-rollback state with the known good state before the initial changes were deployed.

6. Communicate the rollback:

- Inform appropriate stakeholders, including users, about the rollback and the reason for it.
- Provide guidance on any temporary downtime or expected behavior during the rollback process.

7. Post-mortem and lessons learned:

- After the rollback, conduct a post-mortem to understand what went wrong and how to prevent similar issues in the future.
- Document the lessons learned and update relevant processes, procedures, and best practices.

By having a clear rollback plan, you can minimize the impact of any issues that might arise from deploying the changes and ensure a timely and controlled recovery if necessary.

Engineering counterpart
3:02 AM - Responded in 0s

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

Agent response

3:03 AM - Responded in 55s

Concrete next steps with owners and timing:

1. Data validation and testing (Owner: Data Engineer, Due: 2 days)

- Update the `fct\_orders` model to read `region\_code` directly from the `orders` table.
- Create and execute test cases to validate the updated model's output, including:
 - Data type and constraint checks for `region\_code` in the `orders` table.
 - Row count and data distribution checks in the `fct\_orders` model.
 - Comparison of the updated model's output with the previous version or a known good state.
- Ensure that all tests pass before proceeding to the next steps.

2. Dependencies and compatibility check (Owner: Data Engineer & Data Analyst, Due: 3 days)

- Identify and test downstream dependencies that rely on the `fct\_orders` model, such as BI tools, dashboards, and reports.
- Update downstream dependencies as needed to ensure they can handle the changes appropriately.
- Document any changes made to downstream dependencies.

3. Performance benchmarking (Owner: Data Engineer, Due: 2 days)

- Measure the load performance of the updated `fct\_orders` model and compare it with the previous version.
- Ensure that the updated model meets the required service level agreements (SLAs).
- Document the performance results.

4. Backup and rollback plan preparation (Owner: Data Engineer, Due: 1 day)

- Take a backup of the current state before deploying the changes to production.
- Prepare a rollback plan, including:
 - Restoration of the backup.
 - Reversion of the database changes using a version control system or a database migration tool.
 - Reversion of any dependent changes.
 - Monitoring and validation of the rollback.
- Communicate the rollback plan to relevant stakeholders.

5. Regression testing (Owner: Data Engineer & QA Team, Due: 4 days)

- Conduct end-to-end regression testing to ensure that the changes do not introduce any regressions or bugs in other parts of the system.
 - Include testing of data transformation logic, data validation rules, and any downstream processes that rely on the `fct\_orders` model.
- Document any issues found during testing and their resolutions.

6. Monitoring and alerting setup (Owner: Data Engineer, Due: 2 days)

- Set up appropriate monitoring and alerting for the `fct\_orders` model and any downstream dependencies.
- Ensure that the monitoring system can quickly detect and alert on any potential issues that might arise after the changes are deployed.

7. Documentation and communication (Owner: Data Engineer, Due: 1 day)

- Update the relevant documentation to reflect the changes made to the `fct\_orders` model.
- Communicate the changes, including the reasons for the update, the testing performed, and the deployment plan, to appropriate stakeholders, including data engineers, data analysts, and any teams that rely on the `fct\_orders` model.

8. Deployment and verification (Owner: Data Engineer, Due: 1 day)

- Deploy the changes to production, following the deployment process and ensuring minimal downtime.
- Verify the successful deployment by checking the logs and monitoring the system for any potential issues.

9. Post-deployment monitoring (Owner: Data Engineer & On-Call Team, Due: Ongoing)

- Closely monitor the `fct\_orders` model and any downstream dependencies for the first few days after deployment.
- Be prepared to address any user-reported issues or escalations that might arise.

By following these concrete next steps, assigning clear owners, and setting specific timing for each task, you can ensure a well-coordinated and efficient deployment process that minimizes risks and maximizes the chances of a successful update.

Engineering counterpart
3:03 AM - Responded in 0s

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

Agent response

3:03 AM - Responded in 12s

The one thing I'd want a second pair of eyes on is the regression testing process, specifically focusing on the following aspects:

1. Test case coverage:

Ensure that all relevant test cases have been covered, including edge cases, boundary conditions, and any unique business rules.

Verify that the test cases are comprehensive enough to catch any potential regressions or issues introduced by the changes.

A second pair of eyes can help identify any gaps in test case coverage or suggest additional tests to be included.

2. Test data validity:

Validate that the test data used for regression testing is representative and covers a wide range of scenarios.

Ensure that any test data updates or modifications were properly implemented and do not introduce new issues.

A second pair of eyes can help catch any inconsistencies or inaccuracies in the test data.

3. Test environment accuracy:

Confirm that the test environment accurately reflects the production environment in terms of data, configuration, and dependencies.

Ensure that any environment-specific settings or variables have been properly configured for testing.

A second pair of eyes can help identify any discrepancies between the test and production environments that might impact the testing results.

4. Test result interpretation:

Review the test results to ensure that any failures, warnings, or unexpected behaviors have been correctly identified and interpreted.

Verify that the appropriate actions have been taken based on the test results, such as fixing bugs, making adjustments, or documenting known issues.

A second pair of eyes can help catch any misinterpretations or oversights in the test results and provide a different perspective on the findings.

By having a second pair of eyes review the regression testing process, you can increase the confidence in the test results, identify potential blind spots, and minimize the risk of undetected regressions or issues making it to production. This can ultimately lead to a more robust and reliable data platform.

Engineering counterpart
3:03 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.