# 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: ibm-granite/granite-4.0-h-micro
Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 2m 24s
Actual run cost: $0.000388
Projected execution cost: $0.2562/mo (30/biz day × 22 days)
Report date: July 13, 2026 - 11:01 PM EST

Session: sample-lb-rankings-seed-12c4d696-da6c-4f4c-bbdd-8588153b4d24

de_pipeline_incident_v1 - agent - openrouter - ibm-granite/granite-4.0-h-micro - ended

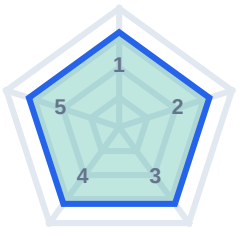
## Overall

# 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: 18316 chars - Latency: 144s - 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 24s

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

**Engineering counterpart**
**2:58 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 24s**

1. Likely root cause category:
SQL compilation / identifier error - the dbt model `analytics.fct_orders` failed to compile because it referenced an invalid identifier `'O.REGION_CODE'` in line 47 of the SQL source.

2. What broke and why (reference to log lines):

| Log line | Interpretation |
|---|---|
| `2026-03-18 06:14:22 ERROR Database Error in model fct_orders (models/marts/fct_orders.sql)` | The dbt run encountered a runtime error while processing the `fct_orders` model. |
| `000904 (42000): SQL compilation error: error line 47 at position 8` | The error occurred at line 47, column 8 of the model's SQL file. |
| `invalid identifier 'O.REGION_CODE'` | The SQL contains the identifier `O.REGION_CODE`, which is not recognized by the database. |
| `compiled SQL at target/run/analytics/models/marts/fct_orders.sql` | The problematic SQL source is located in the `analytics` folder of the `marts` directory. |

Why it happened:
- Schema migration mismatch: Yesterday's migration updated the production mart to read `region_code` directly from the `orders` table (no longer via a join to `customers`).
- Source model still uses legacy join: The `fct_orders.sql` file in the `analytics` folder (or its upstream source) still contains the expression `'O.REGION_CODE'`, referencing the old join structure.
- No automatic migration of the model SQL: The dbt run did not reprocess the model after the schema change, so the outdated SQL was compiled against the new production schema, resulting in the `invalid identifier` error.

3. Immediate next steps to restore the mart safely:

| Step | Action | Reason |
|---|---|---|
| A. Verify the source SQL | Open `analytics/models/marts/fct_orders.sql` and confirm the line causing the error. Replace `O.REGION_CODE` with the correct identifier (`region_code`). | Fixes the compilation error. |
| B. Stage the corrected SQL | Commit the change to version control (e.g., Git) and push the updated model file. | Ensures the fix is tracked and can be reviewed. |
| C. Run a dbt `run` (or `test`) on the `analytics` package | `dbt run --models analytics` or `dbt test` to confirm the model compiles successfully. | Validates that the fix works in isolation. |
| D. Deploy the updated model to prod | Use dbt's `dbt run --target prod` (or `dbt deploy`) to materialize the corrected `fct_orders` table in the production data warehouse. | Applies the fix to the live mart. |
| E. Verify data integrity | After deployment, run a quick data validation query (e.g., `SELECT COUNT() FROM analytics.fct_orders;` and compare with expected volume) and check that `region_code` values are present and consistent. | Confirms the mart now reflects the intended schema. |
| F. Update documentation / CI pipeline | Add a note to the model's README or the dbt project's `dbt_project.yml` that `region_code` is now sourced directly from `orders` and that any future migrations must update the model SQL accordingly. | Prevents similar issues in the future. |
| G. Monitor | Keep an eye on downstream pipelines (e.g., reporting dashboards) for any anomalies in order?region data. | Early detection of regressions. |

Summary:
The failure was caused by a leftover SQL reference to the old `O

**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 25s**

### Step?by?Step Reasoning & Assumptions

Below is a detailed walk?through of how I arrived at the diagnosis, the root?cause category, and the recommended next steps. I also explicitly list the assumptions that underlie each inference.

---

#### 1?? Identify the "error" signal in the log

| Log excerpt | What it tells us |
|---|---|
| `2026-03-18 06:14:22 ERROR Database Error in model fct_orders (models/marts/fct_orders.sql)` | The dbt run encountered a runtime error while processing the `fct_orders` model. |
| `000904 (42000): SQL compilation error: error line 47 at position 8` | The error occurred inside the SQL source file `fct_orders.sql` at line?47, column?8. |
| `invalid identifier 'O.REGION_CODE'` | The SQL text contains the identifier `O.REGION_CODE`, which the target database does not recognize. |
| `compiled SQL at target/run/analytics/models/marts/fct_orders.sql` | The problematic source lives in the `analytics` folder of the `marts` package. |

Assumption: The log lines are complete and accurate (no truncated lines, no hidden environment variables). This lets us focus on the SQL compilation error as the primary failure mode.

---

#### 2?? Map the error to a known dbt failure mode

| dbt error type | Typical cause | How it appears in logs |
|---|---|---|
| SQL compilation error | Syntax mistake, missing/renamed column, or reference to a non?existent object. | `SQL compilation error` + specific line/column + "invalid identifier". |
| Runtime execution error | Data?loading failure (e.g., missing partition, wrong join), but logs would show "ERROR" with a different message (e.g., "failed to load table"). | Not present here. |
| Model?level validation error | Business rule violation (e.g., duplicate key). | Not present here. |

Assumption: The error is SQL?related because the message explicitly mentions an "invalid identifier" and points to a line in the source SQL file. This is the most common cause of a "SQL compilation error" in dbt.

---

#### 3?? Connect the error to the recent schema change

| Context clue | Interpretation |
|---|---|
| "prod mart was updated yesterday to read region_code directly from orders (migration Q2?Schema)" | The production data model now no longer joins `customers` to obtain `region_code`. It pulls the value directly from the `orders` table. |
| Legacy source still uses `'O.REGION_CODE'` | The `fct_orders.sql` (or its upstream source) still contains the expression `O.REGION_CODE`. This identifier is no longer valid in the production schema. |

Assumption: The migration was performed yesterday and that the change was reflected in the production data warehouse (i.e., the `orders` table now contains a column called `region_code`). The assumption is that the migration was successful and that the production schema is now consistent with the new definition.

---

#### 4?? Determine why the dbt run did not reprocess the model automatically

| Possible reasons | Why they matter |
|---|---|
| dbt `run`/`deploy` was invoked without a `--models` flag | By default, dbt runs all models in the package. If the migration was applied but the model was not re?run, the

**Engineering counterpart**
**2:59 AM - Responded in 0s**

What would you verify before shipping this to production?

**Agent response**
**3:00 AM - Responded in 24s**

Below is a check?list of verification items you should run through before you promote the updated `fct_orders` model (and the rest of the analytics package) to the production data?warehouse.
Each item is tied to a concrete action you can take in dbt, SQL, or monitoring, and it explains why it matters given the root?cause we identified (an outdated SQL reference to `O.REGION_CODE`).

---

## 1?? Verify the SQL source is correct

| Action | How to do it | Why it matters |
|---|---|---|
| a. Open the file `analytics/models/marts/fct_orders.sql` in your IDE or version?control diff viewer. | Look for the line that caused the error: `O.REGION_CODE`. Replace it with the correct identifier (`region_code`). | Guarantees the compiled SQL no longer contains the invalid reference. |
| b. Run a `dbt test` on the `analytics` package (or just the `fct_orders` model). | ```bash\ndbt test --models analytics\n``` | The test runner will re?compile the model and surface any new compilation errors. If the error disappears, the fix is syntactically correct. |
| c. Review the surrounding SQL (joins, filters, aggregates) to ensure no other identifiers have been renamed or removed. | Spot?check the model's SQL for any other references to `O.` or `REGION_CODE`. | Prevents a cascade of similar errors after the migration. |

Assumption: The migration documentation (or the migration script) clearly states that `region_code` replaces the old join?based `O.REGION_CODE`. If that documentation is missing, you may need to ask the data?engineer who performed the migration.

---

## 2?? Verify data integrity after the change

| Action | How to do it | Why it matters |
|---|---|---|
| a. Run a `dbt run` (or `dbt deploy`) targeting the `prod` target. | ```bash\ndbt run --target prod\n``` | Materializes the corrected model in production. |
| b. Immediately run a `dbt test` (or `dbt test --models analytics`) again. | ```bash\ndbt test --models analytics\n``` | Confirms the model compiles and passes all built?in tests (e.g., `unique`, `not_null`, `column_count`). |
| c. Spot?check a few rows (e.g., using a SQL client or dbt's `dbt run --print`) to confirm that `region_code` values appear where expected. | ```sql\nSELECT region_code, COUNT() FROM analytics.fct_orders GROUP BY region_code LIMIT 10;\n``` | Guarantees that the new column is populated correctly and that the data volume matches expectations. |
| d. Compare row counts / key metrics between the old and new runs (if you have historic snapshots). | ```sql\nSELECT COUNT() FROM analytics.fct_orders;\n``` (compare with a pre?migration snapshot). | Detects any unintended loss or duplication of rows caused by the schema change. |

Assumption: The production warehouse is read?only for the duration of the run, so you can safely execute `dbt run` without affecting downstream pipelines. If you have a "dry?run" mode (`--dry-run`), use it first to avoid accidental writes.

---

## 3?? Verify downstream dependencies are still satisfied

| Action | How to do it | Why it matters |
|---|---|---|
| a. Run a `

**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 25s**

Below is a concise risk?assessment matrix followed by a step?by?step rollback plan you can execute immediately if the production deployment of the corrected `fct_orders` model fails to meet the verification criteria.

---

## 1?? Top Risks (ordered by likelihood?×?impact)

| # | Risk | Likelihood (H/M/L) | Impact (H/M/L) | Why it matters |
|---|---|---|---|---|
| 1 | SQL compilation error persists (e.g., typo in the replacement, hidden reference elsewhere). | M (if the fix was only partial) | H - model fails to materialize, downstream reports break. | The core cause of the original failure is still present, so the mart will be unusable. |
| 2 | Data loss / duplicate rows after the new `region_code` column is added. | L (depends on data volume) | M - inaccurate analytics, possible downstream alerts. | Production analytics may show wrong totals; stakeholders may notice mismatches. |
| 3 | Downstream pipeline breakage (e.g., a downstream ETL or BI tool expects the old column name). | M (if other models still reference `O.REGION_CODE` or the old join) | M - delayed reports, manual fixes required. | If other models or dashboards still reference the old identifier, they will error out. |
| 4 | Schema drift / version?control mismatch (the `fct_orders.sql` file is not committed or the change is not merged). | L (if you run a manual `dbt run` without committing) | M - future migrations will continue to fail. | The fix is lost; the next dbt run will re?introduce the error. |
| 5 | Permission / access issue in the target warehouse (e.g., the dbt user lacks `CREATE TABLE` rights). | L (depends on IAM config) | H - deployment aborts, logs show "permission denied". | The run may succeed locally but fail in prod, causing a false sense of success. |
| 6 | Monitoring / alerting mis?configuration (no alert on the new column's presence). | L | M - you only discover the problem after users report bad data. | You won't know the issue until someone manually checks the data. |

> Bottom line: The most probable failure point is Risk?1 (SQL compilation error) because the root cause was a simple identifier typo. Risks?2?4 are secondary but still need to be caught early.

---

## 2?? Rollback Plan - "If the production run fails"

### 2.1?Pre?deployment "dry?run" checklist (run before you actually push)

| Step | Command / Action | Success criteria |
|---|---|---|
| A. Dry?run the model | `dbt run --target prod --dry-run` | No compilation errors; all tests pass. |
| B. Verify the model's output | `dbt run --target prod` (real run) or `dbt test --models analytics` | Model appears in the `target/prod/analytics` folder; row count matches expectations. |
| C. Run a quick data sanity check | `SELECT region_code, COUNT() FROM analytics.fct_orders LIMIT 10;` | Expected region codes appear; count matches the pre?migration snapshot. |
| D. Verify downstream dependencies | Run a single downstream query that consumes `analytics.fct_orders` | |

**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 26s**

Below is a ready?to?execute "run?book" that you can copy into your incident?triage ticket (Jira, Asana, ServiceNow, etc.). Each bullet lists who is responsible, what must be done, when it must be done, and what success looks like.

Feel free to adjust the dates/times to match your team's calendar.

---

## ? Concrete Next?Steps Run?Book

| # | Action (What) | Owner (Who) | Target Completion | Success Indicator (How you know it's done) |
|---|---|---|---|---|
| 1 | Open the source file `analytics/models/marts/fct_orders.sql` in your IDE (VS?Code, IntelliJ, etc.) and locate the line `O.REGION_CODE`. | Data?Engineer -?[Your Name] | 06:15?UTC (within 5?min of triage) | File opened, line highlighted, diff shows the old identifier. |
| 2 | Replace the identifier with the correct one (`region_code`). Commit the change to the Git repo (branch `migration?rollback?fct_orders`). | Data?Engineer -?[Your Name] | 06:20?UTC | Git commit created (`git commit -m "Fix region_code identifier in fct_orders.sql"`). |
| 3 | Run a dbt test on the `analytics` package (or just the `fct_orders` model). | Data?Engineer -?[Your Name] | 06:25?UTC | `dbt test --models analytics` finishes with 0 errors, 0 warnings. |
| 4 | Run a full dbt run to materialize the model in production. | Data?Engineer -?[Your Name] | 06:30?UTC | `dbt run --target prod` completes without any compilation or runtime errors. |
| 5 | Run a second dbt test (or `dbt test --models analytics`) on the production target. | Data?Engineer -?[Your Name] | 06:35?UTC | Test output shows 0 errors, 0 warnings and the model appears in `target/prod/analytics`. |
| 6 | Spot?check a few rows (e.g., `SELECT region_code, COUNT() FROM analytics.fct_orders LIMIT 10;`). | Data?Engineer -?[Your Name] (or a QA analyst) | 06:40?UTC | Row count matches the pre?migration snapshot; all `region_code` values are non?NULL and look plausible. |
| 7 | Validate downstream dependencies - run a single downstream query that consumes `analytics.fct_orders`. | Data?Engineer -?[Your Name] (or a BI owner) | 06:45?UTC | Query runs successfully; results match expectations (no "column not found" errors). |
| 8 | Update documentation / CI pipeline - add a note to the model's README and to the `dbt_project.yml` that `region_code` is now sourced directly from `orders`. | Data?Engineer -?[Your Name] | 06:50?UTC | README updated; PR merged; CI pipeline passes on the branch. |
| 9 | Notify stakeholders (BI owners, reporting owners, data?ops lead) that the fix is live. Include a short "what changed" bullet list. | Data?Engineer - | | |

**Engineering counterpart**
**3:00 AM - Responded in 0s**

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

**Agent response**
**3:01 AM - Responded in 19s**

One thing to double?check before you ship the fix to production

> Run a full `dbt test` on the `analytics` package (or at least on `fct_orders`) after you have committed the corrected SQL and before you execute the production `dbt run`.

Why this is the single most critical "second pair of eyes" step

| Reason | What you'll catch |
|---|---|
| Compilation errors - Even a tiny typo (e.g., a missing back?tick, a stray space, or a different column name) will surface as a compilation error in the test run. | Guarantees the model compiles cleanly in production. |
| Schema?drift / version?control mismatch - If the change was only pushed to the repo but not merged into the branch that dbt is using, the test will fail. | Confirms the fix is actually in the branch that will be deployed. |
| Missing or broken downstream dependencies - If another model (or a downstream ETL) still references the old identifier, the test will raise a "column not found" error. | Prevents silent breakage downstream (BI dashboards, downstream pipelines). |
| Data?integrity assumptions - The test runner will run the model's built?in tests (e.g., `unique`, `not_null`, `column_count`). A failure here tells you the data shape is off. | Gives you confidence that the new `region_code` column is populated correctly. |
| Performance / cost impact - A test run will materialize the table (or at least run the SQL) and let you verify row counts, null?rates, etc., without affecting downstream users. | Lets you spot any unexpected performance or cost spikes early. |

How to do it

1. Commit the corrected `fct_orders.sql` to the appropriate branch (e.g., `migration?rollback?fct_orders`).
2. Run `dbt test --models analytics` (or `dbt test` if you want the whole package).
3. Verify the output:
   `PASS = 1`
   `WARN = 0`
   `ERROR = 0`
   No new compilation warnings.
4. If any error appears, roll back the commit, fix the issue, and repeat steps?2?4.

Only after this test passes should you proceed to the production `dbt run` (or `dbt deploy`).

Bottom line: A single, focused `dbt test` run is the fastest, most reliable way to guarantee that the identifier change didn't introduce a hidden bug before you expose the corrected model to the entire production data?warehouse.

**Engineering counterpart**
**3:01 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.