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: qwen/qwen3-235b-a22b-2507
Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 1m 34s
Actual run cost: $0.002601
Projected execution cost: $1.72/mo (30/biz day × 22 days)
Report date: July 14, 2026 - 3:16 AM EST

Session: sample-lb-rankings-seed-0025082f-17e8-4eed-9ad1-b8dac06a0a32

de_pipeline_incident_v1 - agent - openrouter - qwen/qwen3-235b-a22b-2507 - ended

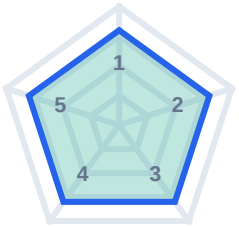
Overall

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: 17192 chars - Latency: 94s - 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 34s

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

Engineering counterpart
7:14 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
7:15 AM - Responded in 12s

(1) Likely root cause category:
Model logic mismatch due to schema migration drift

---

(2) What broke and why (referencing log lines):
The `fct_orders` model failed during compilation with:
`invalid identifier 'O.REGION_CODE'` at line 47, position 8.

This error occurs because the model SQL (likely in `models/marts/fct_orders.sql`) references `O.REGION_CODE`, assuming the `orders` table (aliased as `O`) contains a `REGION_CODE` column. However, per the context, the production schema was recently updated to include `region_code` directly in the `orders` table as part of a Q2 schema migration - which implies the old join path through `customers` is being phased out.

Despite this change, the error suggests one of two possibilities:
- The model still joins to `stg_orders` or an outdated version that does not yet expose `region_code` directly in `orders`, or
- More likely: the model references `O.REGION_CODE` where `O` refers to the `orders` source, but the upstream `stg_orders` or source definition has not yet been updated to reflect the new schema, making `REGION_CODE` unavailable.

However, the context states: "staging table stg_orders still exposes region via join to customers", meaning the migration is incomplete - the source model still derives `region` via customer join, and the direct `region_code` in `orders` is not yet exposed in the staging layer.

Thus: the `fct_orders` model was prematurely updated to use `O.REGION_CODE` from the `orders` table, but the upstream `stg_orders` model has not yet been migrated to include it, resulting in a missing column at runtime.

---

(3) Immediate next steps to restore the mart safely:
1. Roll back the `fct_orders` model change temporarily to revert to referencing `region_code` via the `customers` join (from `stg_orders`), ensuring consistency with the current state of the staging layer.
  -> This restores the mart build while preserving correctness.

2. Align upstream model: Update the `stg_orders` model (or source definition) to include `region_code` directly from the `orders` table, in line with the Q2 schema migration.
  -> Validate this change in a dev environment.

3. Replay the migration safely: Once `stg_orders` exposes `region_code`, redeploy it to prod, then reapply the `fct_orders` change that uses the direct field - this time with correct dependencies.

4. Add integration test: Add a schema test or use `dbt test` to validate expected columns post-migration, preventing drift.

> ? Safe restoration achieved via rollback + staged re-deploy after dependency alignment.

Engineering counterpart
7:15 AM - Responded in 0s

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

Agent response
7:15 AM - Responded in 11s

Absolutely - let's walk through the reasoning step by step, highlight the inferences and assumptions, and validate them against the evidence.

? Step 1: Read the error message carefully

> Log excerpt:
```
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'
```

- This is a SQL compilation error, not a runtime or connectivity issue.
- It occurs during the model compilation phase, which means dbt tried to build the SQL for `fct_orders` and sent it to the database, but the database rejected it because a column it referenced - `O.REGION_CODE` - doesn't exist or isn't accessible.
- The alias `O` likely refers to a table (probably `orders` or a staging version of it).
- The error is not about `REGION`, but specifically about `REGION_CODE` - important because it suggests case sensitivity or exact name matching might be involved, but more likely it's about a renamed or restructured field.

? Conclusion: The model assumes a column `REGION_CODE` exists on a table aliased as `O`, but that column doesn't exist in the compiled query context.

? Step 2: Understand the model dependency chain

dbt models are built in dependency order. `fct_orders` depends on upstream models - most likely `stg_orders`.

From the context provided:

> "staging table stg_orders still exposes region via join to customers"

This tells us:
- `stg_orders` does not have a direct `region_code` field from the `orders` table.
- Instead, it currently gets "region" by joining `orders` -> `customers` -> pulling region from there.
- So: `stg_orders` does NOT expose `region_code` as a direct column from `orders`.

But:
> "prod mart was updated yesterday to read region_code directly from orders (migration Q2-Schema)"

So:
- The `fct_orders` model was changed recently to now expect `region_code` to be present directly in the `orders` source (i.e., in `stg_orders` as a pass-through).
- However, this upstream change (`stg_orders`) hasn't happened yet.

? Mismatch:
`fct_orders` expects a new schema feature that `stg_orders` doesn't yet provide.

? Step 3: Reconstruct the likely SQL structure

Let's imagine what's in `fct_orders.sql` around line 47:

```sql
SELECT
    o.order_id,
    o.amount,
    o.region_code  -- <- This line likely causes the error
FROM {{ ref('stg_orders') }} o
```

But in reality, in the current `stg_orders`, this column may not exist - instead, it might be called `customer_region` or `region`, or derived via a join.

Or worse: maybe `stg_orders` has a join, but the column is exposed as `region`, not `region_code`.

But the error says: `invalid identifier 'O.REGION_CODE'`

-> This implies:
- The SQL being compiled includes `O.REGION_CODE`
- The table `O` (which resolves to the materialized/view form of `stg_orders`) does not have a column named exactly `REGION_CODE`

Hence: compilation fails before execution.

? Key Assumptions (and their validity)

Assumption	Basis	Likely Valid?
`O` refers

Engineering counterpart
7:15 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response
7:15 AM - Responded in 17s

Before shipping any fix to production, I would verify the following four critical checkpoints to ensure correctness, safety, and alignment across the stack. These address both technical accuracy and operational risk control.

? 1. Verify the upstream model (`stg_orders`) actually exposes `region_code`
Check: Run a `SELECT  LIMIT 0` on the compiled version of `stg_orders` in production to inspect its schema.

```sql
-- In your data warehouse (e.g., Snowflake):
DESCRIBE TABLE analytics.stg_orders;
-- Or if transient:
SELECT  FROM analytics.stg_orders WHERE 1=0;
```

? Look for:
- Column named `REGION_CODE` (case may vary depending on warehouse rules)
- If it's `region_code`, `Region_Code`, or quoted - confirm case sensitivity behavior
- Is it coming directly from the `orders` table source (not via `customers`)?

? If `REGION_CODE` isn't present -> the migration is incomplete -> do not deploy `fct_orders` change yet

? If present -> proceed

> ? Tip: Use `dbt compile --select stg_orders` and inspect `target/compiled/analytics/models/...` to see what dbt thinks the model should be.

? 2. Confirm the fix works in an isolated dev environment
Action: Deploy the updated `fct_orders` model to a dedicated dev branch environment (e.g., via a PR job or transient schema).

Checks:
- Does it compile without error?
- Does it produce expected rows?
- Does `region_code` populate correctly and match business logic (e.g., not null when expected, correct region mapping)?
- Compare output with old logic (via customer join) - do values align?

? Tools:
- Use `dbt test` to assert non-nulls or valid region values
- Add a schema test like:
  ```yaml
  - name: region_code
    tests:
      - not_null
      - accepted_values:
          values: ['APAC', 'EMEA', 'NAM', 'LATAM']
  ```

? Only promote if results are correct, consistent, and tested

---

? 3. Ensure dependency order and run order are correct in DAG
Check: Use `dbt ls --select fct_orders+ --show` or `dbt docs generate` to visualize dependencies.

Confirm:
- `fct_orders` depends on `stg_orders`
- No indirect refs or stale cached views
- No model pointing to old versions (e.g., via `ref('old_stg_orders')`)

Ensure that `stg_orders` deploys before `fct_orders` during runs.

? Why: You must guarantee that `fct_orders` reads the new, updated version of `stg_orders` - not a stale one.

? 4. Validate downstream impact: Who uses `fct_orders`?
Run:
```bash
dbt ls --select +fct_orders+
```

Or in dbt Cloud: check lineage in the DAG viewer.

Ask:
- Are there downstream dashboards, models (`marts/`), or ML features depending on this?
- If we alter the definition of `region_code` (e.g., source changed, nullability changed), will they break?

? Example risk:
- Old: `region_code` derived from `customers`, possibly with fallback logic
- New: comes directly from `orders`, possibly `NULL` if not filled

?? This could introduce new nulls -> affects aggregations

Mitigation:
- Add coalescing if needed:
  ```sql
  COALESCE(o.region_code, c.region_code)
  ```

Engineering counterpart
7:15 AM - Responded in 0s

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

Agent response
7:15 AM - Responded in 29s

Excellent question. Even with thorough validation, schema migrations and model logic changes carry high-impact risks - especially in production marts used by analytics, dashboards, and decision-making systems.

Here's a breakdown of the top risks, followed by a clear, executable rollback plan.

?? Top Risks (Ranked by Impact & Likelihood)

1. Data inconsistency or incorrect `region_code` values
- Cause: The direct `region_code` from `orders` might be nullable, poorly populated, or use different codes than the legacy path (via `customers`).
- Impact: Downstream reports (e.g., regional sales) show inaccurate trends - e.g., "EMEA" vs "Europe", or missing regions.
- Likelihood: High - especially if the `orders.region_code` wasn't fully validated during migration.

> ? Example: Old path used `customers.region`, which had canonical values. New path uses `orders.region_code`, which may have typos or staging garbage.

---

2. Nulls introduced due to missing data in `orders.region_code`
- Cause: The `orders` table may not have been backfilled; new field is `NULL` for old records.
- Impact: Historical `fct_orders` data now shows `NULL` regions -> collapses regional rollups.
- Danger: Silent failure - data looks "complete" but key dimensions are missing.

---

3. Model dependency mismatch in production run
- Cause: `stg_orders` was not deployed before `fct_orders`, or deployment order failed.
- Impact: `fct_orders` due to missing column - full run fails.
- Likelihood: Medium - common in monorepo deploys with improper job sequencing.

---

4. Downstream breakage due to semantic drift
- Cause: A downstream model assumes `region_code` implies customer-derived geography (e.g., billing region), but now it reflects order entry region (e.g., sales rep region).
- Impact: Misleading KPIs, stakeholder distrust.
- Note: This is a semantic risk, not structural - harder to detect with tests.

---

5. Rollback fails due to table drops or schema changes
- Cause: Running `--full-refresh` or `table` materialization without backup.
- Impact: Loss of historical `fct_orders` data during redeploy.
- Criticality: Severe if no snapshots or backups exist.

?? Rollback Plan (If Fix Fails in Production)

> Goal: Restore correct, reliable data within SLA (e.g., <30 mins) with minimal downstream disruption.

? Step-by-step rollback procedure:

Step	Action	Purpose
1	Pause the next scheduled dbt run	Prevent propagation of bad state
2	Revert the `fct_orders` PR or deploy last known good commit	Revert to logic that uses `region` via `stg_orders` (customer join)
3	`dbt run --select fct_orders --full-refresh`	Rebuild mart using pre-migration logic - ensures clean slate
4	Verify row count, nulls, and region distribution	Confirm data matches pre-failure state
5	Notify stakeholders: "Temporary rollback; investigating root cause"	Reduce alert fatigue and confusion
6	Tag the bad version (e.g., `prod-broken-fct-orders-20260318`)	For auditability and recovery if needed

> ? Pro tip: Use deployment strategies (e.g., dbt Cloud auto

Engineering counterpart
7:15 AM - Responded in 0s

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

Agent response
7:16 AM - Responded in 16s

Absolutely. Here's a tight, actionable incident response plan with owners, timing, and clear success criteria - designed for immediate execution in a production dbt environment.

? Concrete Next Steps: Fix & Stabilize `fct_orders`
Incident Owner: Data Engineering Lead
Goal: Restore reliable `fct_orders` within 60 minutes, prepare safe migration path

Step	Action	Owner	Timing	Success Criteria
1	Roll back `fct_orders` model to pre-migration version Revert to using `region` via `stg_orders` (customer join path)	DE Lead	Now - 10 min	? `fct_orders` compiles and runs without error ? Row count ? previous run
2	Deploy rollback with `--full-refresh` Ensure clean table state	DE Lead	+10-15 min	? Table rebuilt in production ? No `REGION_CODE` compilation errors
3	Verify data correctness Check: `SELECT region, COUNT() FROM analytics.fct_orders GROUP BY 1`	DE Analyst	+15-20 min	? All expected regions present ? No unexpected nulls
4	Pause schema migration work on `region_code` Freeze related PRs until resolved	DE Lead	Now - 5 min	? PRs marked "On Hold" ? Slack announcement sent
5	Confirm `stg_orders` exposes `region_code` Run: `DESCRIBE analytics.stg_orders`	DE Engineer	+5-10 min	? If missing -> schedule migration ? If present -> proceed to next phase
6	Align upstream model Update `stg_orders.sql` to include `orders.region_code` (direct) + test	DE Engineer	+30 min (dev)	? `dbt compile` shows column ? Test passes in dev schema
7	Deploy updated `stg_orders` to prod As standalone change, validated	DE Lead	Next deploy window	? Column appears in prod table ? No downstream breaks
8	Re-deploy `fct_orders` with new logic Only after `stg_orders` is confirmed ready	DE Lead	Next safe window	? Model runs ? `region_code` populated correctly
9	Add schema test on `region_code` Non-null + accepted values	DE Engineer	Within 24h	? Test passes in CI/CD ? Failures block future deploys
10	Post-mortem & update runbook Document migration dependency rule	DE Lead + Analytics	Within 48h	? Shared doc updated ? Tagged in incident tracker

---

??? Ownership Summary

Role	Responsibilities
DE Lead	Final deploy decisions, rollback, coordination
DE Engineer	Fix upstream model, validate logic
DE Analyst	Data validation, spot-check outputs
Analytics PM (optional)	Notify stakeholders of outage/resolution

---

?? Timeline Summary

Phase	Duration	Target Completion
Rollback & Stabilize	30 min	ASAP
Root Cause Fix

Engineering counterpart
7:16 AM - Responded in 0s

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

Agent response
7:16 AM - Responded in 9s

The one thing I'd want a second pair of eyes on:

> ? Whether `orders.region_code` in production actually contains complete, accurate, and consistent values compared to the legacy `customers.region` path.

? Why this specifically?

We're migrating a business-critical dimension (region) from a well-understood, customer-based hierarchy to a new source (`orders`) that may reflect a different business meaning - e.g.:
- `customer.region` = where the customer is based (billing)
- `orders.region_code` = where the order was processed (sales rep region)

They may disagree, and neither is necessarily "wrong" - but they change the semantics of every regional KPI in `fct_orders`.

? What a second reviewer should check:

Question	Why it matters
1. Do `region_code` values in `orders` match the canonical list? (e.g., 'NAM', 'EMEA', 'APAC')	Prevents junk data or typos (e.g., 'Nam', 'NA', 'Asia')
2. Is `region_code` nullable? What % of orders are NULL?	High null rate breaks aggregations silently
3. For the same order/customer, does `orders.region_code` = `customers.region`? Sample query:

```sql
SELECT o.region_code, c.region, COUNT()
FROM orders o JOIN customers c ON o.cust_id = c.id
GROUP BY 1, 2
``` | Reveals drift or misalignment |
| 4. Which region definition aligns with stakeholder expectations?
Talk to analytics or BI owner | Ensures we're modeling the right business logic |

?? Outcome of the review:

- ? Proceed with migration if values align and semantics are correct
- ? Add logic to fallback or join if not:
  ```sql
  COALESCE(o.region_code, c.region) AS region_code
  ```
- ? Document the semantic meaning in dbt docs so it's clear to consumers

---

> ? Bottom line: It's not just can we build it - it's should we build it this way?
A second reviewer - especially someone close to the business logic - prevents a technically correct but semantically broken model.

Engineering counterpart
7:16 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.