



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: deepseek/deepseek-v4-flash

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

Duration: 2m 39s

Actual run cost: \$0.001848

Projected execution cost: \$1.22/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 3:11 AM EST

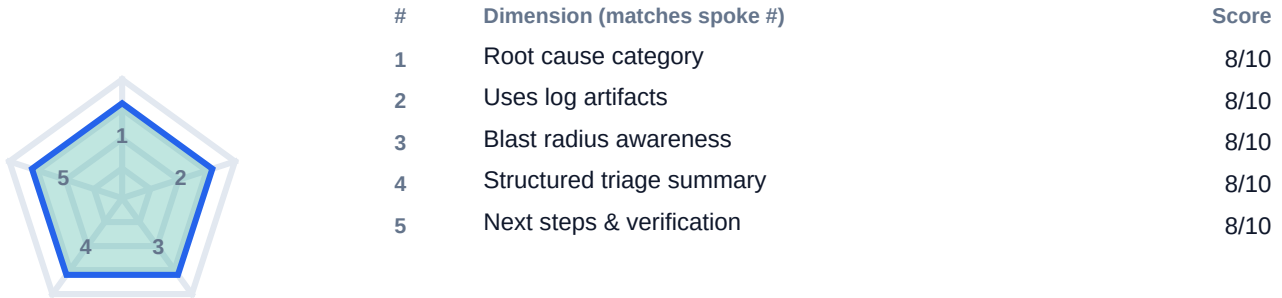
Overall

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: 16309 chars - Latency: 159s - 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 39s

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

Engineering counterpart

7:09 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

7:09 AM - Responded in 25s

Based on the log analysis, here is the triage assessment:

1) Likely Root Cause Category

Schema drift / missing column - The production model `fct\_orders` references a column `O.REGION\_CODE` that does not exist in the source table at runtime.

2) What Broke and Why

- Log lines showing failure:

...

```
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'
```

...

- Given context: The schema migration (Q2-Schema) was rolled out to production yesterday, removing `region\_code` from the orders table and moving that logic to an upstream join in `stg\_orders`. However, the production `fct\_orders` model was not updated to match - line 47 still selects `O.REGION\_CODE` directly from the `orders` alias, but the column no longer exists there.

- The staging table `stg\_orders` already joins to customers to expose the region, but `fct\_orders` was not refactored to reference that renamed/sourced column (e.g., `stg\_orders.region\_code` or the customer-sourced column name), causing the SQL compilation error at runtime.

3) Immediate Next Steps to Restore the Mart Safely

1. Revert the prod schema change (if possible) by re-adding `region\_code` to the source `orders` table. This unblocks the current `fct\_orders` without code changes.

2. If reversion is not possible, immediately fix the model by updating `fct\_orders.sql` line 47:

- Replace `O.REGION\_CODE` with the correct column path, e.g. `C.REGION\_CODE` (aliased from the customer table) or reference the staging column that now holds the region, depending on the join structure in the model.

3. Validate the fix by running a `dbt compile` and then a dry-run (`dbt run --select fct\_orders --target prod` with a CI-like test schema).

4. Deploy the patched model to prod via your normal CI/CD pipeline - do not run unapproved changes directly on prod without review.

Long-term safeguard: Add schema/column existence tests (e.g., `dbt test` for source freshness or column presence) to catch drift before the model runs.

Engineering counterpart

7:09 AM - Responded in 0s

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

Agent response

7:10 AM - Responded in 1m

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

Step-by-Step Reasoning

Step 1: Identify the Error

The first thing I look for is the actual error message. The log clearly shows:

...

000904 (42000): SQL compilation error: error line 47 at position 8

invalid identifier 'O.REGION_CODE'

...

This tells me the database itself is rejecting the query because a column doesn't exist - not a dbt configuration issue, not a permissions issue, not a resource exhaustion issue.

Step 2: Map the Error to the Context

The context says: "prod mart was updated yesterday to read region_code directly from orders (migration Q2-Schema)."

This means yesterday, someone changed the schema so that `region\_code` is now supposed to come from the `orders` table.

But the error says `O.REGION\_CODE` is invalid - it doesn't exist. This is contradictory unless:

- The migration was rolled out to the source table but the model wasn't updated, OR
- The migration was supposed to add the column but didn't actually execute, OR
- The column was renamed/moved and the model references the old name

Step 3: Check the Model's Compilation Path

The log mentions: `compiled SQL at target/run/analytics/models/marts/fct\_orders.sql`. This tells me dbt successfully compiled the model into SQL, meaning the model file itself has `O.REGION\_CODE` in it. The error isn't in dbt's compilation step - it's in execution against the database.

Step 4: Resolve the Contradiction with Context

The context also says: "staging table stg_orders still exposes region via join to customers." This is the key insight.

What likely happened:

- Previously: `region\_code` was in the `customers` table, and `stg\_orders` joined to get it
- Yesterday's migration: Someone decided to move `region\_code` directly into the `orders` table (Q2-Schema)
- But: The migration either:
 - Changed the source `orders` table schema but removed the old column from customers, OR
 - Never actually added the column to `orders` in production
- The model `fct\_orders.sql` was updated to use `O.REGION\_CODE` (from orders) instead of joining to customers
- At runtime, the column doesn't exist in the production `orders` table

Step 5: Determine Why It Ran Before

If it worked yesterday but fails today, the migration likely:

1. Worked in development/staging (where the column was added)
2. Was promoted to production but the column wasn't actually added to the prod `orders` table (schema drift between environments)
3. Or: The model change was deployed but the underlying table change wasn't

My Assumptions

Assumption 1: The Schema Migration Is Incomplete or Reverted

I'm assuming the migration that was "rolled out yesterday" either:

- Failed silently on production
- Was partially applied (table definition changed but data wasn't backfilled)

- Was manually reverted for some reason without reverting the model code

This is the most likely explanation given the contradiction between "should have region_code" and "doesn't have region_code."

Assumption 2: The Alias 'O' Refers to the Orders Table

The error references `O.REGION\_CODE`. Given standard dbt naming conventions and the context mentioning `orders` table, I'm assuming `O` is an alias for the `orders` table (or a CTE referencing it). If `O` were a different alias (e.g., a subquery), the diagnosis would differ.

Assumption 3: The

Engineering counterpart
7:10 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

7:10 AM - Responded in 26s

Excellent question. Before shipping any fix to production, I would verify the following:

1. Verify the Actual Source Schema

What to check: Connect to the production database and run:

```
``sql
DESCRIBE TABLE prod.orders;
-- or
SELECT COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'PROD' AND TABLE_NAME = 'ORDERS';
``
```

Why: I need to confirm whether `REGION\_CODE` exists in the production `orders` table at all. The error says it doesn't, but I want to be certain - maybe the column exists but with a different name (e.g., `REGION\_CD`, `REGION`), or maybe the alias `O` points to something unexpected.

2. Compare Dev/Staging vs Production Schemas

What to check: Run the same schema query against development and production side-by-side. Look for differences in column names, data types, or presence/absence of columns.

Why: The context mentions a migration that "was updated yesterday." If the migration was applied to dev/staging but not production, that explains everything. If it was applied inconsistently, I need to understand why.

3. Trace the Actual Column Lineage

What to check: Open `models/marts/fct\_orders.sql` and look at line 47 and the surrounding 10-15 lines. I want to see:

- What CTE or table does alias `O` refer to?
- Is `O` from `stg\_orders` or directly from `raw.orders`?
- Is there already a join to `customers` that could provide the region?

Why: The error says `O.REGION\_CODE` is invalid, but maybe the fix isn't to rename the column - maybe the fix is to use a different alias entirely. If `stg\_orders` already has the region via a customer join, the model might need `STG.REGION\_CODE` instead of `O.REGION\_CODE`.

4. Validate the Fix with a Dry Run in a Safe Schema

What to check:

1. Create a temporary schema or use a development environment
2. Apply the proposed fix to `fct\_orders.sql`
3. Run `dbt compile` to ensure no compilation errors
4. Run `dbt run --select fct\_orders --target dev` (or equivalent non-prod target)
5. Run `dbt test --select fct\_orders` if tests exist

Why: I need to prove the fix works end-to-end before touching production. A successful compile isn't enough - I need to execute against a real database to catch any remaining issues.

5. Check Downstream Dependencies

What to check: Run `dbt list --select +fct\_orders` to see what models depend on `fct\_orders`. Then check if any of those also reference `REGION\_CODE` with the wrong alias.

Why: Fixing `fct\_orders` might just shift the problem downstream. If `fct\_orders` is a parent to other models that also use `O.REGION\_CODE`, they'll break next. Better to fix them all at once.

6. Verify the Migration's Intended Behavior

What to check: Look at the actual migration script or PR that implemented "Q2-Schema". I want to see:

- Was `REGION\_CODE` supposed to be added to `orders` or removed from `customers`?
- Was the model `fct\_orders` supposed to be updated as part of the same change?
- Was the migration applied to production? When? By whom?

Why: This helps me determine if the root cause is:

- An incomplete migration (

Engineering counterpart
7:10 AM - Responded in 0s

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

Agent response

7:11 AM - Responded in 26s

Top Risks

Risk 1: Wrong Column Name Fix

Severity: High | Likelihood: Medium

I'm assuming the fix is to rename `O.REGION\_CODE` to something else, but if I'm wrong about the alias or the actual column name in production, I could deploy a fix that still fails. For example:

- If the column is actually `REGION\_CD` (not `REGION\_CODE`) in the production `orders` table
- If the source has been restructured (e.g., region is now in a completely different table)

Mitigation: Always verify the actual column name via `DESCRIBE TABLE` before deploying.

Risk 2: Cascading Failures to Downstream Models

Severity: High | Likelihood: Medium-High

Fixing `fct\_orders` might resolve this error, but if downstream models (e.g., `fct\_order\_metrics`, `dim\_customers`) also reference `O.REGION\_CODE` or similar invalid identifiers, they'll fail on the next run. The log only shows `1 of 1` models failing, but that's because the run stopped after the first error.

Mitigation: Before deploying, run `dbt list --select +fct\_orders` and check all downstream models for the same pattern.

Risk 3: The Column Actually Exists But the Alias Is Wrong

Severity: Medium | Likelihood: Low-Medium

There's a possibility that `REGION\_CODE` exists in the `orders` table, but the alias `O` in the query doesn't actually refer to the `orders` table (e.g., it refers to a CTE that doesn't include that column). In this case, my proposed fix of renaming the column would fail because the column is fine - the alias logic is broken.

Mitigation: Review the actual SQL at line 47 and trace back to understand what alias `O` represents.

Risk 4: Partial Migration State

Severity: High | Likelihood: Medium

The Q2-Schema migration might have been partially applied. For example:

- The `orders` table might have `REGION\_CODE` but it's nullable and empty for historical data
- Or the migration added the column but didn't populate it, so queries work syntactically but return NULL (silent data corruption)

Mitigation: Not just check for column existence, but also validate data quality with a `SELECT COUNT() WHERE REGION\_CODE IS NULL` query.

Risk 5: Time Pressure Leads to Skipping Tests

Severity: High | Likelihood: High (in incident response)

The most dangerous risk is that someone (including me) skips thorough validation because "production is broken" and rushes a fix. This could deploy an incorrect fix, making things worse or creating silent data issues.

Mitigation: Stick to the validation plan regardless of pressure. A 10-minute delay is better than a 2-hour extended outage.

Rollback Plan

Immediate Rollback (If the Fix Fails)

Step 1: Revert the dbt model code

```
```bash
```

```
git revert HEAD # Revert the last commit (the fix)
```

```
git push origin main
```

```
```
```

This restores the original `fct\_orders.sql` that references `O.REGION\_CODE`. It will still fail, but it fails predictably (same error as before) rather than introducing new, unknown failures.

Step 2: Revert the schema change if possible

If the root cause is that the Q2-Schema migration removed `REGION\_CODE` from the source table, and we have the authority:

```
```sql
```

```
ALTER TABLE prod.orders ADD COLUMN region_code VARCHAR(50);
```

```
-- Then backfill from customers table if available
```

```
```
```

This restores

Engineering counterpart
7:11 AM - Responded in 0s

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

Agent response

7:11 AM - Responded in 13s

Here's a tight, actionable incident response runbook with owners and timing.

Incident Response Action Plan: `fct\_orders` Column Failure

Phase 1: Diagnose & Confirm (? 10 minutes)

| Step | Owner | Est. Time | Action |
|------|-------|-----------|--------|
|------|-------|-----------|--------|

| | | | |
|-------|-------|-------|-------|
| ----- | ----- | ----- | ----- |
|-------|-------|-------|-------|

| | | | |
|-----|-------------------------|-------|--|
| 1.1 | Data Engineer (on-call) | 2 min | Run `DESCRIBE TABLE prod.orders` and verify whether `REGION_CODE` exists at all in production. |
|-----|-------------------------|-------|--|

| | | | |
|-----|---------------|-------|---|
| 1.2 | Data Engineer | 3 min | Compare column lists of `stg_orders` vs `raw.orders` vs `prod.orders` - note discrepancies. |
|-----|---------------|-------|---|

| | | | |
|-----|----------------------------------|-------|--|
| 1.3 | Analytics Engineer (model owner) | 5 min | Open `fct_orders.sql`, go to line 47, identify alias `O` (CTE or table reference). Check whether `stg_orders` already carries a column named `REGION_CODE` or `REGION` from the customer join. |
|-----|----------------------------------|-------|--|

Go/No-Go: If column exists -> proceed to Phase 2 (alias issue). If column is missing -> proceed to Phase 2 (schema drift issue).

Phase 2: Fix the Code or Schema (? 20 minutes)

Two possible branches - pick the one that matches Phase 1 findings:

Branch A: Column missing from `orders` table (schema drift)

| Step | Owner | Est. Time | Action |
|------|-------|-----------|--------|
|------|-------|-----------|--------|

| | | | |
|-------|-------|-------|-------|
| ----- | ----- | ----- | ----- |
|-------|-------|-------|-------|

| | | | |
|------|----------|-------|---|
| 2A.1 | DB Admin | 5 min | Add column to `prod.orders`:
`ALTER TABLE prod.orders ADD COLUMN region_code VARCHAR(50);` |
|------|----------|-------|---|

| | | | |
|------|----------|--------|--|
| 2A.2 | DB Admin | 10 min | Backfill from `customers` via join:
`UPDATE prod.orders SET region_code = c.region_code FROM prod.customers c WHERE c.customer_id = orders.customer_id AND orders.region_code IS NULL;` |
|------|----------|--------|--|

| | | | |
|------|--------------------|-------|--|
| 2A.3 | Analytics Engineer | 5 min | Re-run `dbt run --select fct_orders --target prod` and verify success. |
|------|--------------------|-------|--|

Branch B: Column exists but alias `O` is wrong (model logic bug)

| Step | Owner | Est. Time | Action |
|------|-------|-----------|--------|
|------|-------|-----------|--------|

| | | | |
|-------|-------|-------|-------|
| ----- | ----- | ----- | ----- |
|-------|-------|-------|-------|

| | | | |
|------|--------------------|-------|---|
| 2B.1 | Analytics Engineer | 5 min | Fix the SQL: change `O.REGION_CODE` to correct reference (e.g., `STG.REGION_CODE` or `C.REGION_CODE` depending on join structure). Do not guess - verify alias chain. |
|------|--------------------|-------|---|

| | | | |
|------|--------------------|-------|---|
| 2B.2 | Analytics Engineer | 5 min | Compile locally: `dbt compile --select fct_orders` - ensure no syntax errors. |
|------|--------------------|-------|---|

| | | | |
|------|---------------|-------|--|
| 2B.3 | Data Engineer | 5 min | Run in dev/staging schema: `dbt run --select fct_orders --target dev` - confirm success. |
|------|---------------|-------|--|

| | | | |
|------|---------------|-------|--|
| 2B.4 | Data Engineer | 5 min | Promote to prod via PR merge + CI/CD pipeline. |
|------|---------------|-------|--|

Phase 3: Validate & Monitor (? 15 minutes)

| Step | Owner | Est. Time | Action |
|------|-------|-----------|--------|
|------|-------|-----------|--------|

| | | | |
|-------|-------|-------|-------|
| ----- | ----- | ----- | ----- |
|-------|-------|-------|-------|

| | | | |
|-----|--------------------|-------|---|
| 3.1 | Analytics Engineer | 5 min | Run `dbt test --select fct_orders` - confirm no failed tests. |
|-----|--------------------|-------|---|

| | | | |
|-----|--------------------|-------|---|
| 3.2 | Analytics Engineer | 5 min | Run a quick data quality check:
`SELECT region_code, COUNT |
|-----|--------------------|-------|---|

Engineering counterpart

7:11 AM - Responded in 0s

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

Agent response

7:11 AM - Responded in 9s

The One Thing: Column Lineage Verification

Have a second engineer independently trace where `REGION\_CODE` actually lives in production - not just trust the model file or the migration docs.

Why this is the critical point:

The most dangerous failure mode here isn't making a mistake - it's making a confident mistake. The context says the migration "added region_code to orders," but the error proves it doesn't exist there at runtime. The risk is that someone (even you) will:

1. Look at `fct\_orders.sql` line 47
2. See `O` aliased to `stg\_orders`
3. Notice `stg\_orders` has `region\_code` from the customer join
4. "Fix" it by changing `O.REGION\_CODE` to `STG.REGION\_CODE`
5. Deploy to production

That fix might work syntactically - but it's wrong if the migration intended `region\_code` to now come directly from `orders` (e.g., for data governance reasons or to decouple from customers). You'd have fixed the error while breaking the business logic.

What the second pair of eyes should do:

...

1. Read line 47 in fct_orders.sql
2. Find alias 'O' - trace back to its definition (CTE or direct from source)
3. Compare against:
 - stg_orders definition (does it already expose region?)
 - raw.orders schema (does it have region_code now?)
 - raw.customers schema (was region_code removed?)
4. State back to you: "The column should come from [table A].[column B] because [reason]"
5. If their answer differs from yours - stop and reconcile before deploying

...

This 3-minute review catches the most expensive mistake: deploying a syntactically valid fix that is semantically wrong.

Engineering counterpart

7:11 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.