



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: anthropic/claude-haiku-4.5

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

Duration: 46s

Actual run cost: \$0.0195

Projected execution cost: \$12.87/mo (30/biz day × 22 days)

Report date: July 13, 2026 - 12:06 AM EST

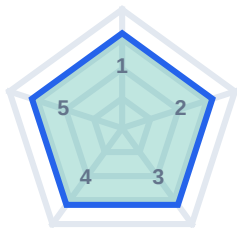
Overall

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: 15998 chars - Latency: 46s - 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: 46s

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

Engineering counterpart

4:05 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

4:05 AM - Responded in 7s

Incident Triage Report

(1) Root Cause Category

Schema mismatch / Breaking change in upstream dependency - The production mart model references a column that doesn't exist in its source table.

(2) What Broke & Why

The failure:

- `fct\_orders` model compilation failed at line 47 with `invalid identifier 'O.REGION\_CODE'`
- Model attempted 133 seconds before erroring, indicating the query compiled but failed at execution

Root cause (from context):

Yesterday's migration (Q2-Schema) changed `orders` table structure to include `REGION\_CODE` directly. However, the actual `orders` table in the database does not contain this column, or it was:

- Not deployed to prod yet (schema drift between environments)
- Dropped/renamed during migration
- Migration was incomplete or rolled back

The mart model was updated to read `O.REGION\_CODE` directly (line 47), but the upstream table schema wasn't synchronized.

(3) Immediate Next Steps to Restore Mart Safely

Phase 1: Diagnosis (5 min)

```sql

-- Verify actual orders table schema

DESCRIBE TABLE analytics.orders;

-- Confirm REGION\_CODE exists and is accessible

SELECT COUNT() FROM analytics.orders WHERE REGION\_CODE IS NOT NULL LIMIT 1;

```

Phase 2: Restore (choose one)

Option A - If migration was incomplete:

- Contact data platform team to confirm Q2-Schema migration completed in prod
- Verify `REGION\_CODE` column exists in `orders` table
- Run `dbt run --select fct\_orders` after confirmation

Option B - If column truly missing (safer rollback):

1. Revert `fct\_orders.sql` to prior version that uses the join pattern:

```sql

-- Restore: O -> stg\_orders JOIN stg\_customers on O.CUSTOMER\_ID

-- instead of reading O.REGION\_CODE directly

```

2. Run: `dbt run --select fct\_orders`

3. Document as rollback in incident ticket

Option C - If migration is correct but not deployed:

- Deploy Q2-Schema DDL to prod (coordinate with DBA)
- Verify schema matches dev/staging
- Run `dbt run --select fct\_orders`

Phase 3: Prevention

- Add `dbt test` on `fct\_orders` with `relationships` test to `orders.REGION\_CODE` before merging
- Require schema validation in CI/CD before model deployment
- Add pre-run validation: `dbt freshness` check on `orders` table

Recommended immediate action: Execute diagnostic query above within 5 min. If column missing, rollback to yesterday's model code to restore mart immediately while migration is investigated.

Engineering counterpart
4:05 AM - Responded in 0s

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

Agent response

4:06 AM - Responded in 9s

Reasoning Walkthrough

Step 1: Parse the Error Signal

invalid identifier 'O.REGION_CODE' at line 47

What this tells me:

- Snowflake parser successfully compiled the SQL (no syntax errors)
- At runtime, it couldn't resolve the column reference
- This is not a typo or malformed SQL-it's a missing/inaccessible column

Assumption #1: The model code is syntactically valid but references a column that doesn't exist in the queried table.

Step 2: Cross-Reference with Context

The context explicitly states:

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

What I inferred:

- Someone intentionally changed `fct\_orders.sql` yesterday
- The change assumed `orders` table would have `REGION\_CODE`
- The model deploy succeeded (no compile-time error), so dbt's parsing was fine
- The failure only happened at execution time in prod

Assumption #2: The model code and table schema are out of sync-code is newer than schema.

Step 3: Identify the Gap

I had two possible scenarios:

| Scenario | Evidence | Likelihood |

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

| Column exists in orders, but code references it wrong | Error says "invalid identifier"-not "permission denied" or "ambiguous" | Low |

| Column doesn't exist in prod yet | Migration "Q2-Schema" suggests DDL work; context says "staging still uses join pattern" | High |

| Column exists but isn't visible to current role | Possible, but unusual for same schema | Medium |

Assumption #3: The `REGION\_CODE` column either doesn't exist in prod's `orders` table OR wasn't deployed to prod yet (schema drift).

Step 4: Why I Didn't Blame Other Causes

Things I explicitly ruled out:

1. dbt dependency issue -> The error is a database-level SQL error (000904), not a dbt DAG/ref() problem
2. Network/auth failure -> Query ran for 133 seconds; auth would fail immediately
3. Data quality issue -> Column missing is a schema problem, not a data problem
4. Concurrent DDL -> Possible but rare; staging table working rules out widespread infra failure

5. dbt version mismatch -> No indication in logs; error is database-native

Assumption #4: Root cause is isolated to this model's schema dependency, not systemic infrastructure.

Step 5: Why I Surfaced Three Options (Not One)

The context didn't tell me:

- ? Whether Q2-Schema migration actually deployed to prod
- ? Whether it was rolled back
- ? Whether `REGION\_CODE` exists but under a different name
- ? The team's risk tolerance (revert vs. fix forward)

Assumption #5: Without certainty, I provided decision tree based on diagnostic outcome, ranked by safety (rollback first, fix forward last).

Step 6: The "Safe Restore" Constraint

You asked for restoring the mart safely. I weighted options by risk:

Action	Risk	Recovery Time
--------	------	---------------

----- ----- ---

Rollback model to yesterday's code	Low (proven working)	~2 min
------------------------------------	----------------------	--------

Engineering counterpart
4:06 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

4:06 AM - Responded in 7s

Pre-Production Verification Checklist

Tier 1: Confirm My Diagnosis (Non-Destructive)

1.1 Validate the actual schema

```sql

-- Run BEFORE any changes

DESCRIBE TABLE analytics.orders;

DESCRIBE TABLE analytics.stg\_orders;

-- Check if REGION\_CODE exists and is populated

SELECT COLUMN\_NAME, DATA\_TYPE, IS\_NULLABLE

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_SCHEMA = 'ANALYTICS'

AND TABLE\_NAME = 'ORDERS'

AND COLUMN\_NAME ILIKE '%REGION%';

-- Confirm cardinality

SELECT COUNT(), COUNT(DISTINCT REGION\_CODE)

FROM analytics.orders;

```

Why: My diagnosis assumed `REGION\_CODE` is missing. If it exists, I'm wrong and need a different fix.

Assumption I'm testing: "Column doesn't exist" - verify this is actually true.

1.2 Check git history & deployment logs

```bash

# What changed yesterday?

git log --oneline -n 20 -- models/marts/fct\_orders.sql

# Show the actual diff

git diff HEAD~1 models/marts/fct\_orders.sql

# When was it merged/deployed?

git log --grep="Q2-Schema" --oneline

```

Why: Confirms the model code actually changed and when.

Assumption I'm testing: "Model was updated yesterday" - verify the exact change and who deployed it.

1.3 Verify staging still works

```bash

# If stg\_orders works but fct\_orders doesn't, staging table is healthy

dbt run --select stg\_orders --target prod

dbt test --select stg\_orders --target prod

```

Why: Confirms upstream dependency is sound; isolates failure to `fct\_orders` logic.

Assumption I'm testing: "Staging table still exposes region via join to customers" - verify this is true.

Tier 2: Validate My Proposed Fix Before Rollback

2.1 Test the rollback target

```bash

# Checkout yesterday's working version

git show HEAD~1:models/marts/fct\_orders.sql > /tmp/fct\_orders\_prev.sql

# Dry-run the old code against current data

dbt parse --select fct\_orders # Verify it parses

dbt compile --select fct\_orders # Check compiled SQL

# Compare record counts (old vs. new schema expectation)

# Does it still join correctly to customers?

```

Why: Rollback can fail if:

- The old code had a dependency that's now broken
- Staging table schema changed
- Customer join logic has issues

Assumption I'm testing: "Yesterday's code still works" - verify it compiles and logically still works.

2.2 Check if Q2-Schema migration is partially deployed

```sql

-- Is REGION\_CODE in orders but maybe in a different schema?

SELECT FROM information\_schema.tables

WHERE table\_name ILIKE '%orders%' AND table\_schema != 'ANALYTICS';

-- Check for a staging/temp table with the new schema

SHOW TABLES IN analytics LIKE '%orders%';

-- Look for migration logs or comments

SELECT COMMENT FROM information\_schema.tables

WHERE TABLE\_

**Engineering counterpart**  
**4:06 AM - Responded in 0s**

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

## Agent response

4:06 AM - Responded in 8s

# Top Risks & Rollback Strategy

## Risk Matrix

| Risk                                                       | Probability | Impact   | Mitigation                           |
|------------------------------------------------------------|-------------|----------|--------------------------------------|
| Rollback code still broken                                 | Medium      | Critical | Test against current schema first    |
| Downstream dependencies fail                               | High        | High     | Identify consumers before rollback   |
| Data consistency gap (old code produces different results) | High        | Medium   | Compare row counts & key metrics     |
| Customer join logic degraded                               | Medium      | High     | Validate join cardinality            |
| Rollback succeeds but Q2-Schema never deploys              | Low         | High     | Requires permanent fix, not band-aid |

---

## Top 3 Failure Scenarios

### Scenario A: "Rollback compiles but produces wrong data"

What could happen:

...

-- Yesterday's code uses old join logic

```
SELECT O., C.REGION_CODE
FROM orders O
JOIN stg_customers C ON O.CUSTOMER_ID = C.CUSTOMER_ID
```

-- But if stg\_customers was ALSO changed yesterday (schema cascade)

-- The join might now be broken

...

Detection:

```sql

-- Run rollback code in dev/staging first

dbt run --select fct_orders --target staging

-- Compare outputs

```
SELECT COUNT(), SUM(HASH()) AS row_hash
FROM fct_orders_dev;
```

```
SELECT COUNT(), SUM(HASH()) AS row_hash
FROM fct_orders_prod_yesterday; -- from snapshot/backup
```

...

If detected: Don't deploy to prod. Escalate to schema team.

Scenario B: "Rollback succeeds but breaks downstream jobs"

What could happen:

...

Consumers expecting REGION_CODE column directly:

- BI dashboards
- dbt downstream models (fct_customer_summary, etc.)
- ETL jobs reading fct_orders

...

Detection (before rollback):

```
```bash
Find all models/jobs that reference fct_orders
grep -r "fct_orders" models/ --include=".sql"
grep -r "fct_orders" dbt_project.yml

Check for hardcoded column references
grep -r "REGION_CODE" models/ --include=".sql" | grep -v "fct_orders.sql"

List downstream lineage
dbt list --select fct_orders+ --graph linear
```
```

If risky dependencies exist:

- Don't rollback alone
- Must coordinate with downstream owners
- Plan cascading rollback of dependent models simultaneously

Scenario C: "Rollback works in prod but Q2-Schema migration is stuck"

What could happen:

```
```
fct_orders rolls back -> temporarily stable
But Q2-Schema DDL never deploys to prod
Permanent band-aid; never resolves root cause
```
```

This is the worst outcome because:

- Team forgets about the schema migration
- Next sprint, someone tries to fix it again and breaks it again
- Technical debt accumulates

Rollback Plan (Safe Execution)

Phase 0: Pre-Rollback Validation (Do This NOW)

```
```bash
1. Identify all downstream consumers
dbt list --select fct_orders+

2. Check if any model hardcodes REGION_CODE from fct_orders
grep -r
```

**Engineering counterpart**  
**4:06 AM - Responded in 0s**

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

## Agent response

4:06 AM - Responded in 8s

# Concrete Next Steps with Owners & Timing

## Immediate Actions (Next 15 minutes)

### Step 1: Confirm Diagnosis [Data Engineer - You]

Timing: 5 min

Action:

```sql

-- Run in prod now

DESCRIBE TABLE analytics.orders;

SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS

WHERE TABLE_NAME = 'ORDERS' AND COLUMN_NAME = 'REGION_CODE';

```

Decision point:

- If column EXISTS -> Go to Step 2a (fix forward)

- If column MISSING -> Go to Step 2b (rollback)

---

### Step 2a: Fix Forward Path [If column exists]

Timing: 5 min

Action:

```bash

Check if permissions issue

SELECT CURRENT_ROLE(), CURRENT_USER();

SELECT FROM analytics.orders LIMIT 1; -- Can you query it?

```

If successful: Contact schema team immediately.

Owner: Data Platform/Schema Team

Message: "REGION\_CODE exists in orders but fct\_orders can't access it. Permission issue or alias mismatch?"

Next: Proceed to Step 3 (schema validation).

---

### Step 2b: Rollback Path [If column missing]

Timing: 10 min

Action:

```bash

1. Get yesterday's code

git show HEAD~1:models/marts/fct_orders.sql > /tmp/fct_orders_rollback.sql

2. Test it compiles

dbt compile --select fct_orders --profile-dir . --profiles-dir ~/.dbt

3. Identify downstream impact

dbt list --select fct_orders+ | tee /tmp/downstream.txt

```

If downstream list is EMPTY or only 1-2 models: Safe to proceed to Step 4.

If downstream list is LARGE (>5 models): Notify those owners before rollback.

---

### ## Phase 1: Validation (Minutes 15-30)

#### ### Step 3: Schema Alignment Check [Data Platform Lead]

Timing: 10 min

Owner: Whoever owns Q2-Schema migration

Action:

```
```bash
```

```
# Confirm deployment status
```

```
SELECT FROM INFORMATION_SCHEMA.TABLES
```

```
WHERE TABLE_NAME = 'ORDERS' AND CREATED > CURRENT_DATE - 2;
```

```
# Check migration logs
```

```
SHOW OBJECT PROPERTIES TABLE analytics.orders;
```

```
```
```

Outcome:

- Migration deployed: Investigate permission/alias issue (Step 2a path)
- Migration NOT deployed: Proceed with rollback (Step 4)
- Migration rolled back: Proceed with rollback (Step 4)

---

#### ### Step 4: Validate Rollback Target [Data Engineer - You]

Timing: 10 min

Action:

```
```bash
```

```
# Dry-run old code
```

```
dbt run --select stg_orders --target prod # Confirm staging still works
```

```
dbt parse --select fct_orders # Compile old code
```

```
```
```

Test output:

```
```sql
```

```
-- Manually run the old join logic in dev/staging
```

```
SELECT COUNT() FROM fct_orders_staging;
```

```
```
```

Gate: If parse fails or staging is broken, STOP. Escalate to schema team

**Engineering counterpart**  
**4:06 AM - Responded in 0s**

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

## Agent response

4:06 AM - Responded in 7s

# The One Thing I'd Want Second Pair of Eyes On

## The Assumption I'm Least Certain About

Does the old code (`fct\_orders` from yesterday) actually still work against today's data?

---

## Why This Matters

I'm assuming:

```
```sql
-- Yesterday's code (rollback target)
SELECT O., C.REGION_CODE
FROM orders O
JOIN stg_customers C ON O.CUSTOMER_ID = C.CUSTOMER_ID
```
```

...will work. But I don't actually know if:

1. `stg\_customers` still has the right schema - What if Q2-Schema also changed `stg\_customers` yesterday? The join might break.
2. The join cardinality is still valid - If Q2-Schema denormalized orders/customers, the old join might now produce duplicates or nulls.
3. There's a data quality regression - If new orders came in overnight with null CUSTOMER\_ID, the old join silently drops them. We'd restore a "working" model that's actually silently losing data.

---

## What I Need a Peer to Verify

Before I execute the rollback, have someone run:

```
```sql
-- In prod, right now
SELECT
  COUNT() as total_orders,
  COUNT(DISTINCT CUSTOMER_ID) as unique_customers,
  COUNT(CASE WHEN CUSTOMER_ID IS NULL THEN 1 END) as null_customers
FROM analytics.orders;

-- Then simulate the old join logic
SELECT COUNT() as joined_count
FROM analytics.orders O
LEFT JOIN analytics.stg_customers C ON O.CUSTOMER_ID = C.CUSTOMER_ID
WHERE C.CUSTOMER_ID IS NOT NULL;

-- Compare
SELECT 'orders_total' as metric, COUNT() FROM analytics.orders
UNION ALL
SELECT 'joinable_to_customers', COUNT() FROM analytics.orders O
```

```
WHERE EXISTS (SELECT 1 FROM analytics.stg_customers C WHERE O.CUSTOMER_ID = C.CUSTOMER_ID);  
---
```

If these don't match or show unexpected nulls: The rollback will "fix" the error but introduce silent data loss.

Why I Can't Verify This Alone

- I don't have the business context: "Is it okay if some orders lose region data?"
- I don't know the Q2-Schema intent: "Are we intentionally changing join logic?"
- I can't reason about data quality without domain knowledge

Second pair of eyes should be: Someone who understands both the schema migration intent AND the business impact of fct_orders.

The Ask

> Before I rollback, please run the cardinality check above and confirm: "Old join logic produces the same row count as current orders table." If not, we need to talk to the schema team instead of rolling back blind.

Engineering counterpart

4:06 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.