



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-chat

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

Duration: 1m 57s

Actual run cost: \$0.004477

Projected execution cost: \$2.95/mo (30/biz day × 22 days)

Report date: July 13, 2026 - 10:58 PM EST

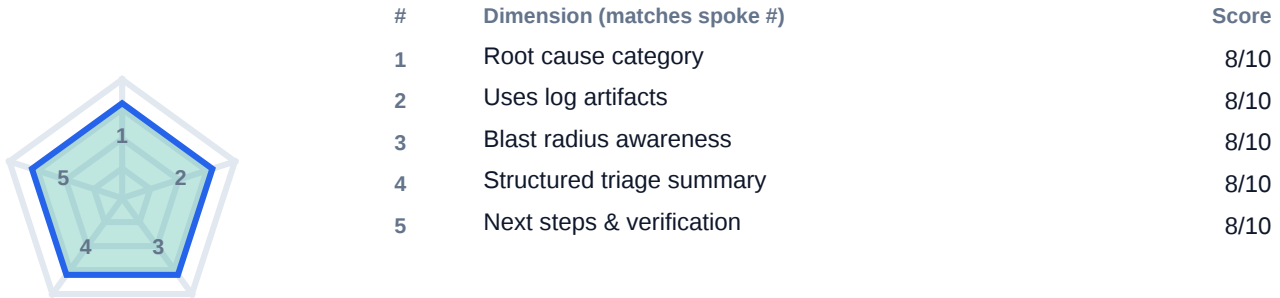
Overall

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: 16090 chars - Latency: 117s - 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 57s

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

Engineering counterpart

2:56 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

2:56 AM - Responded in 18s

Incident Triage Report: Failed dbt Production Run

(1) Root Cause Category

Schema/Column Reference Error (upstream schema change not properly synchronized with dependent models)

(2) What Broke and Why

- Failed Model: `fct\_orders` (critical mart table)
- Direct Error:

```
...
```

```
SQL compilation error: invalid identifier 'O.REGION_CODE'
```

```
...
```

The model references `REGION\_CODE` in the `orders` table (`O` alias), but this column either:

- Does not exist in the source table (due to a schema change).
 - Was renamed/moved in the source but not updated in the model.
- Context Clue:
- The log notes a recent migration (`Q2-Schema`) to read `region\_code` directly from `orders`, but the staging table (`stg\_orders`) still joins to `customers` for region data. This suggests a schema drift between the source (`orders`) and the model's expectations.

(3) Immediate Next Steps

1. Rollback (Short-Term Fix):

- Revert the `fct\_orders` model to use the prior logic (joining to `customers` for region) until the schema migration is confirmed complete.

- Example:

```
```sql
```

```
-- Replace O.REGION_CODE with C.REGION (or prior logic)
```

```
SELECT ..., C.REGION AS region_code
```

```
FROM {{ ref('stg_orders') }} O
```

```
JOIN {{ ref('dim_customers') }} C ON O.customer_id = C.customer_id
```

```
```
```

2. Validate Upstream Schema (Long-Term Fix):

- Check the `orders` table schema in production to confirm if `REGION\_CODE` exists.
- If missing: Coordinate with the team responsible for the `Q2-Schema` migration to ensure the column is added.

3. Add Safeguards:

- Add a `pre-hook` or test to validate critical column existence before model execution.
- Example test:

```
```sql
```

```
-- schema.yml
```

```
models:
```

```
- name: fct_orders
```

```
tests:
```

```
- dbt_utils.expression_is_true:
```

```
expression: "{{ adapter.get_columns_in_relation(ref('stg_orders')) | map(attribute='column') | list |
```

```
includes('region_code') }}"
```

```
error_message: "REGION_CODE missing in stg_orders!"
```

```
```
```

Priority: High (blocking mart refresh).

Expected Resolution Time: 15-30 minutes for rollback; 1-2 days for schema alignment.

Key Takeaway: Schema migrations require coordinated updates to downstream models. Use tests to catch drift early.

Engineering counterpart
2:56 AM - Responded in 0s

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

Agent response

2:57 AM - Responded in 24s

Step-by-Step Reasoning and Assumptions

1. Parsing the Logs

- Observed Error:

...

SQL compilation error: invalid identifier 'O.REGION_CODE'

...

- This indicates the SQL engine couldn't find `REGION\_CODE` in the table aliased as `O` (`orders`).
- The error is not a timeout, permission issue, or syntax error-it's a missing column.

- Context Line:

...

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).

...

- This suggests:

- Previously, `region` data came from a join to `customers`.
- A migration (`Q2-Schema`) changed this to expect `REGION\_CODE` directly in `orders`.

2. Key Assumptions

1. Schema Drift:

- The `orders` table was supposed to gain a `REGION\_CODE` column as part of `Q2-Schema`, but:
- The column might not have been deployed yet.
- Or, it was deployed under a different name (e.g., `region\_id`, `region`).

2. Model Logic Change:

- The `fct\_orders` model was updated to use `O.REGION\_CODE` (directly from `orders`), but the upstream table doesn't match this expectation.

3. Staging Table Still Uses Old Logic:

- `stg\_orders` joins to `customers` for region data, meaning the raw `orders` table might not have `REGION\_CODE` yet.

4. Deployment Timing Issue:

- The migration (`Q2-Schema`) and model update were not synchronized.
- Likely scenario:
- The `fct\_orders` model was updated to use `REGION\_CODE` before the `orders` table was altered.

3. Validating Assumptions

To confirm, I'd check:

- Source Schema:

```
```sql
```

```
-- Run in prod database
```

```
SELECT column_name
```

```
FROM information_schema.columns
```

```
WHERE table_name = 'orders' AND table_schema = 'raw';
```

```
```
```

- If `REGION\_CODE` is missing, the migration wasn't applied correctly.

- Model SQL:

- Look at `models/marts/fct\_orders.sql` to verify if it switched from joining to `customers` to using `O.REGION\_CODE`.

4. Why This is the Most Likely Root Cause

- The error is deterministic (compilation, not runtime).

- The log explicitly mentions a schema migration (`Q2-Schema`) and conflicting logic.
- No other errors (e.g., timeouts, connectivity) suggest infrastructure issues.

5. Alternative Explanations (Ruled Out)

- Typo in SQL: Unlikely, since the model compiled successfully in dev/test (would fail earlier).
- Permissions: Would throw a `ACCESS DENIED` error, not `invalid identifier`.
- Data Volume/Performance: The model failed during compilation, not execution.

6. Immediate Actions Justification

- Rollback First:
 - The mart is broken; reverting to the working logic (joining to `customers`) restores functionality.
 - Faster than debugging the migration.
- Schema Validation Next:
 - Ensure the `orders` table has the expected column before re-attempting the

Engineering counterpart
2:57 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

2:57 AM - Responded in 27s

Before shipping the fix for `fct\_orders` to production, I would verify the following to ensure a safe deployment:

1. Verify Upstream Schema State

Check if `REGION\_CODE` exists in source tables

```
```sql
```

-- In production database:

```
SELECT
```

```
 column_name
```

```
FROM
```

```
 information_schema.columns
```

```
WHERE
```

```
 table_name IN ('orders', 'stg_orders')
```

```
 AND table_schema = 'raw';
```

```
```
```

- If `REGION\_CODE` is missing in `orders`, the migration (`Q2-Schema`) was not applied correctly.
- If `stg\_orders` still uses `region` from `customers`, confirm backward compatibility.

Why?

- Ensures the model's assumptions about column existence are correct.
- Prevents "invalid identifier" errors in production.

2. Validate the Fix in a Non-Prod Environment

Test the updated `fct\_orders` model in staging:

- If rolling back to the old logic (joining to `customers`):

```
```sql
```

```
SELECT
```

```
 ...,
```

```
 C.region AS region_code -- Revert to old column
```

```
FROM
```

```
 {{ ref('stg_orders') }} O
```

```
JOIN
```

```
 {{ ref('dim_customers') }} C ON O.customer_id = C.customer_id
```

```
```
```

- If proceeding with `REGION\_CODE` (after confirming its existence):

```
```sql
```

```
SELECT
```

```
 ...,
```

```
 O.region_code -- New column
```

```
FROM
```

```
 {{ ref('stg_orders') }} O
```

```
```
```

Verify:

- The model builds successfully.
- Row counts match expectations (no data loss).
- Aggregations (e.g., `SUM(revenue)`) are consistent with historical values.

Why?

- Catches SQL errors before they hit production.
- Ensures data consistency post-change.

3. Check Downstream Dependencies

Identify impacted dashboards/reports:

```
```sql
```

-- Use dbt docs or query lineage metadata:

```
SELECT
```

```
 dependent_model
```

```
FROM
```

```
 dbt_lineage
```

```
WHERE
```

```
 upstream_model = 'fct_orders';
```

```
```
```

- Notify owners of dependent assets (e.g., BI teams) about the change.
- Confirm no dashboards hardcode `region\_code` logic (unlikely, but worth checking).

Why?

- Avoids breaking downstream consumers if column semantics change (e.g., `region` vs. `region\_code`).

4. Review Migration Timing

Coordinate with the `Q2-Schema` migration owner:

- Was `REGION\_CODE` added to `orders`? If not, when?
- If the column is coming later, ensure the model logic aligns with the rollout plan.

Why?

- Prevents future breaks if the model expects a column that doesn't exist yet.

5. Add Safeguards

Add tests to catch schema drift:

```
```yaml
```

```
models/schema.yml
```

```
models:
```

```
 - name: fct_orders
```

```
 tests:
```

```
 - dbt_utils.expression_is_true:
```

```
 expression: "{{ adapter.get_columns_in_relation(ref('stg_orders')) | map(attribute='column') | list | includes('region') }}"
```

```
 error_message: "Required column 'region' missing in stg_orders!"
```

```
```
```

Why?

- Fails fast if upstream schemas change unexpectedly.

6. Deployment Plan

If reverting to old logic:

- Merge the fix (join to `customers`) and deploy.
- Monitor the next production run for errors.

If `

Engineering counterpart
2:57 AM - Responded in 0s

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

Agent response

2:57 AM - Responded in 13s

Top Risks

1. Schema Mismatch:
 - If `REGION\_CODE` doesn't exist in `orders`, the model will fail again.
 - Risk Level: High (blocks mart refresh).
2. Data Inconsistency:
 - Reverting to `region` from `customers` might introduce mismatches if the data semantics differ from `region\_code`.
 - Risk Level: Medium (BI reports might show incorrect results).
3. Downstream Breakage:
 - Dashboards or other models relying on `fct\_orders` might break if column names or data types change.
 - Risk Level: Medium (dependent teams may need to adjust queries).
4. Migration Timing Issues:
 - If `REGION\_CODE` is added later, deploying the fix now could cause future breaks.
 - Risk Level: Medium (requires coordination with migration team).
5. Test Coverage Gaps:
 - If tests don't catch schema drift, similar issues could occur in future migrations.
 - Risk Level: Low (mitigated by adding schema tests).

Rollback Plan

Scenario 1: Model Fails After Deployment

1. Identify Failure:
 - Monitor logs for errors like `invalid identifier` or `column not found`.
2. Revert Code:
 - Roll back the `fct\_orders` model to the previous working version (using `region` from `customers`).
 - Example:

```
```sql
SELECT ..., C.region AS region_code
FROM {{ ref('stg_orders') }} O
JOIN {{ ref('dim_customers') }} C ON O.customer_id = C.customer_id
```
```
3. Re-deploy:
 - Merge and deploy the rollback commit.
 - Confirm the model runs successfully in production.
4. Communicate:
 - Notify stakeholders (e.g., BI teams) about the rollback.

Scenario 2: Data Issues Detected After Deployment

1. Identify Issue:
 - Compare `region` vs. `region\_code` values in `fct\_orders` to ensure consistency.
2. Rollback:
 - Revert the model to the previous version (as above).

3. Fix Data:

- Investigate discrepancies and correct data in the `orders` or `customers` table if needed.

4. Re-deploy:

- Once data is fixed, re-deploy the corrected model.

Scenario 3: Downstream Dashboards Break

1. Identify Breakage:

- Check logs or notify BI teams to confirm dashboard failures.

2. Revert Code:

- Roll back the `fct\_orders` model to the previous version.

3. Coordinate with Consumers:

- Work with downstream teams to update their queries or reports.

4. Re-deploy:

- Deploy a fix for `fct\_orders` after coordinating with consumers.

Mitigation Steps

1. Backup:

- Ensure backups of `fct\_orders` exist before deploying changes.

2. Dry Run:

- Test the fix in staging or a non-production environment before deploying to production.

3. Monitoring:

- Set up alerts for model failures or data discrepancies post-deployment.

4. Documentation:

- Document the rollback process and share it with the team for future incidents.

Key Takeaway

The primary risk is schema mismatch, which can be mitigated by validating upstream tables and

Engineering counterpart
2:57 AM - Responded in 0s

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

Agent response

2:58 AM - Responded in 23s

Concrete Next Steps with Owners & Timing

1. Immediate Fix (0-30 mins)

Owner: Data Engineer (you)

Action:

- Rollback `fct\_orders` to the previous working version (using `customers.region`).
- Verify in Staging: Confirm the model builds successfully.
- Deploy to Prod: Merge and trigger a production run.

Check:

```
```sql
-- Verify rollback logic
SELECT C.region AS region_code
FROM {{ ref('stg_orders') }} O
JOIN {{ ref('dim_customers') }} C ON O.customer_id = C.customer_id
```
```

2. Validate Upstream Schema (30-60 mins)

Owner: Data Engineering + Migration Team

Action:

- Check `orders` table in production:

```
```sql
-- Run in prod DB
SELECT column_name
FROM information_schema.columns
WHERE table_name = 'orders' AND column_name = 'REGION_CODE';
```
```

- If missing: Escalate to the migration owner to confirm `Q2-Schema` rollout timeline.

Outcome:

- Column exists: Proceed with updating `fct\_orders` to use `O.REGION\_CODE`.
- Column missing: Delay changes until migration completes.

3. Coordinate with Downstream (1-2 hrs)

Owner: Analytics/BI Team

Action:

- Notify stakeholders (Slack/email):
 - > "`fct\_orders` rolled back to use `customers.region`. Dashboards relying on `region\_code` logic may need updates."
- Verify critical dashboards:
 - Check if any hardcoded references to `region\_code` exist.

4. Add Safeguards (Next 24 hrs)

Owner: Data Engineer

Action:

- Add schema tests to `stg\_orders`:

```
```yaml
tests:
```

```
- dbt_utils.expression_is_true:
 expression: "{{ 'REGION_CODE' in adapter.get_columns_in_relation(ref('stg_orders')) | map(attribute='column') | list
}}"
 error_message: "REGION_CODE missing in stg_orders!"
 ...
```

- Document the incident in runbooks:
  - > "Schema migrations require: (1) Column existence checks, (2) Staging/prod sync."

---

#### #### 5. Final Deployment (After Migration Confirmed)

Owner: Data Engineer + Migration Team

Action:

- Update `fct\_orders` to use `O.REGION\_CODE` once the column is confirmed in prod.
- Monitor for 24 hrs to ensure no downstream breaks.

---

#### ### Rollback Plan (If Issues Post-Deployment)

1. Revert `fct\_orders` SQL to the join-based logic.
2. Trigger a manual prod run to restore the mart.
3. Post-mortem: Identify why safeguards failed.

---

#### ### Timeline Summary

Step	Owner	Timeframe
-----	-----	-----
Rollback `fct_orders`	Data Engineer	0-30 mins
Validate `orders.REGION_CODE`	Migration Team	30-60 mins
Notify downstream teams	BI Team	1-2 hrs
Add tests/documentation	Data Engineer	24 hrs
Final deploy (if applicable)	Data Engineer	Post-migration

Key: Prioritize restoring the mart first, then prevent recurrence

### Engineering counterpart

2:58 AM - Responded in 0s

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

### Agent response

2:58 AM - Responded in 12s

### The One Thing to Double-Check: Upstream Schema State

Ask a teammate to verify:

1. Does `REGION\_CODE` exist in the production `orders` table?

```
```sql
```

```
-- Have them run in prod:
```

```
SELECT
```

```
  EXISTS (
```

```
    SELECT 1
```

```
    FROM information_schema.columns
```

```
    WHERE table_schema = 'raw'
```

```
    AND table_name = 'orders'
```

```
    AND column_name = 'REGION_CODE'
```

```
  ) AS has_region_code;
```

```
```
```

- If yes: Proceed with updating `fct\_orders` to use it.

- If no: Confirm the migration (`Q2-Schema`) timeline with the migration owner.

2. Is `stg\_orders` still joining to `customers` for `region`?

- Check the staging model's SQL to ensure backward compatibility.

Why This Matters Most:

- The root cause hinges on whether `REGION\_CODE` was truly added to `orders`. A second pair of eyes prevents false assumptions.

- Misdiagnosing this could lead to another production failure.

Bonus: If the teammate confirms the column is missing, escalate to the migration owner immediately to align on timelines.

Template for Asking:

> "Hey [teammate], can you quickly verify if `REGION\_CODE` exists in `raw.orders` in prod? The model broke expecting it, but I suspect the migration hasn't landed yet. Here's the query to run: [above SQL]."

This cuts through ambiguity and ensures the fix is based on reality, not guesses.

### Engineering counterpart

2:58 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.