



Vantage RuntimeAI

Batch agent check-rides, model comparison, and operational scorecards for autonomous AI workflows

8.4/10

Overall score

Run summary

Scenario: Lineage Impact Review

Model: google/gemma-4-31b-it

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

Duration: 3m 18s

Actual run cost: \$0.001614

Projected execution cost: \$0.3551/mo (10/biz day × 22 days)

Report date: July 14, 2026 - 12:06 AM EST

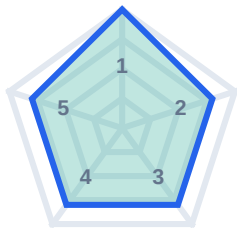
Overall

8.4/10

PASS - trust=high

Pass - score 8.4/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



#	Dimension (matches spoke #)	Score
1	Downstream impact mapped	10/10
2	References lineage artifact	8/10
3	Migration / cutover risk	8/10
4	Ordered impact summary	8/10
5	Safe migration plan	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.

- Legacy column named: Met
- Multiple downstream marts covered: Met
- Migration plan for unmigrated rows: Met
- Rollback / compat considered: 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: 16958 chars - Latency: 198s - Criteria: 4/4

Score analysis

Maps impact across multiple downstream marts.

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. 4 of 4 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

- + Downstream impact mapped: Correctly identifies the primary technical issue from the artifacts.
- + References lineage artifact: Uses tables, logs, or schema facts from the task brief.
- + Migration / cutover risk: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Ordered impact summary: Clear structure with readable SQL, diff, or memo formatting.
- + Safe migration plan: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Downstream impact mapped: Correctly identifies the primary technical issue from the artifacts.
- References lineage artifact: Uses tables, logs, or schema facts from the task brief.
- Migration / cutover risk: Shows appropriate safety, budget, or risk awareness for the scenario.
- Ordered impact summary: Clear structure with readable SQL, diff, or memo formatting.
- Safe migration plan: 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

Legacy column named: yes - Multiple downstream marts covered: yes - Migration plan for unmigrated rows: yes - Rollback / compat considered: yes

CAVEATS

- Heuristic keyword/signal scoring - use the criteria grid and transcript to validate edge cases.

Transcript

Session

Scenario: Lineage Impact Review

Agent response <-> Engineering counterpart

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

Duration: 3m 18s

Engineering counterpart

4:03 AM

Task brief - Lineage impact review (complex complexity)

Engineering proposes dropping a legacy column. Assess downstream impact:

```
```json
{
 "proposed_change": {
 "action": "DROP COLUMN",
 "table": "raw.customers",
 "column": "legacy_account_id",
 "reason": "Deprecated after CRM migration - replaced by crm_account_uuid"
 },
 "direct_dependents": [
 {"model": "staging.stg_customers", "column_refs": ["legacy_account_id"]},
 {"model": "intermediate.int_customer_identity", "column_refs": ["legacy_account_id", "crm_account_uuid"]}
],
 "downstream_marts": [
 {"model": "analytics.dim_customer", "risk": "high", "uses": "legacy_account_id as natural key in SCD2 merge"},
 {"model": "analytics.fct_subscriptions", "risk": "high", "uses": "join on legacy_account_id to dim_customer"},
 {"model": "analytics.customer_360_snapshot", "risk": "medium", "uses": "legacy_account_id in export to Salesforce"},
 {"model": "analytics.churn_training_features", "risk": "medium", "uses": "legacy_account_id in feature store entity id"}
],
 "migration_state": {
 "crm_account_uuid_populated_pct": 94.2,
 "legacy_only_accounts": 1204
 },
 "deadline": "2026-07-12 (CRM cutover)"
}
```
```

Agent response

4:04 AM - Responded in 1m 37s

Impact Review: Deprecation of `raw.customers.legacy\_account\_id`

Review Status: ? BLOCKING (High Risk)

Senior Engineer Note: The proposed drop is currently unsafe. While the migration is 94.2% complete, the dependency on `legacy\_account\_id` as a natural key for SCD2 (Slowly Changing Dimensions) and joining logic in core marts creates a high probability of data loss and broken reporting if executed as a hard drop.

1. Affected Marts & Model Analysis

| Model | Impact Level | Technical Risk | Consequence of Drop |

| :--- | :--- | :--- | :--- |

| `analytics.dim\_customer` | ? Critical | SCD2 Merge Key | Dropping the natural key will break the merge logic, potentially creating duplicate records or failing to track historical changes for existing customers. |

| `analytics.fct\_subscriptions` | ? Critical | Join Key | Total loss of relational integrity. Subscriptions will fail to join to the customer dimension, resulting in "Unattributed" or NULL customers in all revenue reporting. |

| `analytics.customer\_360\_snapshot` | ?? Medium | External Export | Salesforce sync will fail if the receiving API expects the legacy ID for record matching/upserting. |

| `analytics.churn\_training\_features` | ?? Medium | Feature Store ID | ML models trained on legacy IDs will be unable to generate inferences for current users, causing a "cold start" or failure in the production pipeline. |

2. Migration Risks for Unmigrated Accounts

Current Gap: 5.8% of records (1,204 accounts) lack a `crm\_account\_uuid`.

Orphaned Data: These 1,204 accounts will become "ghost records." Because the join key is gone, any historical subscription or event data tied to these accounts will be effectively deleted from the analytical layer.

Identity Fragmentation: In `int\_customer\_identity`, the logic likely handles the mapping between legacy and UUID. Without the legacy ID, we lose the ability to resolve identities for the remaining 5.8% of the user base.

SCD2 Corruption: Since `dim\_customer` uses this as a natural key, the system cannot determine if a record is an "update" or a "new insert" for these specific accounts.

3. Recommended Migration Order

A "Hard Drop" is not advised. I recommend a three-phase deprecation strategy:

Phase 1: Logic Migration (Immediate)

1. Update `int\_customer\_identity`: Implement a COALESCE logic: `COALESCE(crm\_account\_uuid, legacy\_account\_id) AS unified\_account\_id`.
2. Refactor Marts: Update `dim\_customer` and `fct\_subscriptions` to join on the new `unified\_account\_id`.
3. Backfill: Force a full refresh of `dim\_customer` to ensure the new key is propagated across all historical snapshots.

Phase 2: Soft Deprecation (Observation)

1. Rename to `deprecated\_legacy\_account\_id`: Rename the column in `raw.customers` (if the DB supports it) or create a view.
2. Monitor: Log any queries still hitting the legacy column via query history/audit logs.
3. Cleanse: Resolve the 1,204 missing UUIDs via a data cleanup script before proceeding.

Phase 3: Hard Drop (Post-

Engineering counterpart
4:04 AM - Responded in 0s

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

Agent response

4:05 AM - Responded in 22s

To arrive at the "Blocking" status, I performed a dependency chain analysis. Instead of looking at the column as a piece of data, I looked at it as a functional anchor for the entire data warehouse.

Here is the step-by-step reasoning and the assumptions that drove my conclusions.

Step 1: Analyzing the "Gravity" of the Column

I first looked at where the column sits in the pipeline. It is in ``raw.customers``. In a modern data stack, the raw layer is the "single source of truth." If you delete data here, you cannot "undo" it without a backup restore.

I then looked at the Direct Dependents. Since it flows into ``stg_customers`` $\rightarrow$ ``int_customer_identity``, any break at the raw level creates a cascading failure. If the staging model fails to compile because a column is missing, every single downstream mart (the "leaf nodes") stops updating.

Step 2: Evaluating the "Structural" Risk (The SCD2 Problem)

The most critical red flag was the note: ``legacy_account_id`` as natural key in SCD2 merge for ``dim_customer``.

My reasoning:

An SCD2 (Slowly Changing Dimension Type 2) table tracks history by creating new rows for every change. To know which row to expire and which to insert, the system needs a Natural Key (a unique identifier that never changes).

If you remove that key, the merge logic (e.g., ``MERGE INTO dim_customer USING stg_customers ON dim_customer.id = stg_customers.id``) fails.

Result: The pipeline will either crash or, worse, treat every single existing customer as a "new" customer, duplicating your entire customer base in the dimension table.

Step 3: Calculating the "Data Loss" Gap

I looked at the migration state: ``crm_account_uuid_populated_pct: 94.2%``.

My reasoning:

94% sounds high, but in data engineering, the risk lives in the 6%.

1,204 accounts are "legacy only." If I drop the legacy ID today, those 1,204 accounts have zero identifiers.

When ``fct_subscriptions`` tries to join to ``dim_customer`` using the new UUID, those 1,204 accounts will result in ``NULL`` joins.

Result: You lose visibility into the revenue and activity of ~6% of your customers instantly.

Step 4: Assessing External Integration (The "Silent" Failure)

Finally, I looked at the ``customer_360_snapshot`` (Salesforce export).

My reasoning:

Internal warehouse failures are loud (the pipeline crashes). External failures are silent (the data just disappears from Salesforce).

If Salesforce uses the ``legacy_account_id`` to match records, the export will start creating duplicate accounts in the CRM or failing to update existing ones.

Assumptions I Made

Since I don't have access to the actual SQL code, I made the following professional assumptions based on the provided JSON:

1. Assumption on Join Logic: I assumed that ``fct_subscriptions`` does not currently have the ``crm_account_uuid`` stored in its own raw table and relies entirely on the join to ``dim_customer`` to resolve identity.
2. Assumption on SCD2 Implementation: I assumed the SCD2 merge is performed using a ``MERGE`` or ``UPSERT`` statement where the ``legacy_account_id`` is the primary join predicate.

3.

What would you verify before shipping this to production?

Agent response

4:05 AM - Responded in 27s

Before shipping this change to production, I would require a Verification Gate consisting of four specific technical validations. I would not approve the PR until the following evidence is provided:

1. The "Null-Join" Audit (Quantifying Data Loss)

Before dropping the column, we must prove that moving to the new UUID results in zero data loss.

The Test: Run a "Left Join" query between the current production `fct\_subscriptions` and the proposed new `dim\_customer` (using the new UUID).

The Success Criteria: The number of orphaned records (where the join is NULL) must be zero, or exactly match the known 1,204 accounts that we have explicitly agreed to archive/ignore.

Why: This proves that the 5.8% gap has been resolved or accounted for.

2. SCD2 Idempotency Test (The "Dry Run")

Since this is a natural key change, we must ensure the history isn't corrupted.

The Test: Restore a production backup of `dim\_customer` into a staging environment. Run the new migration logic (the merge using `crm\_account\_uuid`) against it.

The Success Criteria:

Compare the row count before and after. It should not spike unexpectedly.

Verify that a single customer's history (their timeline of changes) remains intact and hasn't been split into two different identities.

Why: This prevents the "Duplicate Customer" catastrophe.

3. Downstream Contract Validation (The "API Check")

We need to verify the "silent failures" in external systems.

The Test: Coordinate with the Salesforce/CRM admin to check the External ID settings on the Account object.

The Success Criteria: Confirmation that Salesforce has been updated to accept `crm\_account\_uuid` as the primary Upsert key, or that a mapping table exists in the middleware.

Why: This ensures the `customer\_360\_snapshot` doesn't break the CRM integration.

4. Lineage "Ghost" Search (The "Grepping" Phase)

The provided JSON is a snapshot, but documentation is often outdated.

The Test: Perform a global search (grep) across the entire codebase (dbt project, Airflow DAGs, Looker/Tableau YAMLS, and stored procedures) for the string `"legacy\_account\_id"`.

The Success Criteria: Every single instance found must be mapped to one of the models listed in the impact review. Any "hidden" dependencies (e.g., a random analyst's ad-hoc view) must be migrated.

Why: To avoid the "Monday Morning Surprise" where a critical executive dashboard breaks because it was using a column we thought was unused.

Final Checklist for the "Go" Decision:

| |
|---|
| Requirement Status Verifier |
| :--- :---: :--- |
| \$text{UUID Populated} = 100\%\$ [] Data Quality Audit |
| SCD2 Merge Validated [] QA/Staging Run |
| SFDC Key Updated [] CRM Admin |
| Global Code Search Clean [] Lead Engineer |
| Rollback Script Tested [] DevOps/DE |

Engineering counterpart
4:05 AM - Responded in 0s

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

Agent response

4:05 AM - Responded in 25s

Top Risks: The "Nightmare Scenarios"

If this is deployed without the mitigations discussed, these are the three most likely critical failures:

1. The "Identity Split" (Highest Probability)

The Risk: Because 5.8% of accounts lack a UUID, the system will treat them as "new" entities.

Impact: A customer who has been with the company for 5 years suddenly appears as a brand-new user in the dashboard. Their entire historical spend, churn risk, and subscription history vanish from their profile and move to an "Unknown/NULL" bucket.

2. The "SCD2 Explosion" (Highest Severity)

The Risk: The merge logic fails to find a match for existing records because the natural key is gone.

Impact: Instead of updating existing rows, the pipeline inserts a new row for every single customer every time the model runs. Your `dim\_customer` table grows exponentially, causing query timeouts and reporting massive (incorrect) growth in customer counts.

3. The "CRM Feedback Loop" (External Impact)

The Risk: The Salesforce export fails or creates duplicates.

Impact: Sales reps see duplicate accounts in the CRM. If the export is used to trigger automated emails or billing updates, customers may receive duplicate notifications or incorrect invoices.

Rollback Plan: The "Safety Net"

A `DROP COLUMN` operation is destructive. You cannot simply "undo" it with a SQL command. Therefore, the rollback plan is tiered based on how the change was implemented.

Scenario A: The "Hard Drop" (Worst Case)

If the column was deleted directly from the raw table.

Rollback Action: Point-in-Time Recovery (PITR). Restore the `raw.customers` table from a database snapshot taken immediately before the deployment.

Recovery Time Objective (RTO): High (Hours). Requires DB admin intervention and potential downtime for the raw ingestion pipeline.

Risk: Any data ingested between the drop and the restore will be lost unless re-synced from the source.

Scenario B: The "View-Based Deprecation" (Recommended)

If we replaced the raw table with a view that excludes the column.

Rollback Action: View Redefinition. Update the view definition to include `legacy\_account\_id` again.

Recovery Time Objective (RTO): Low (Seconds). A simple `CREATE OR REPLACE VIEW` statement.

Risk: Virtually zero.

Scenario C: The "Logical Drop" (dbt/Transformation layer)

If the column exists in raw but was removed from the staging/mart models.

Rollback Action: Git Revert. Revert the commit in the dbt repository and trigger a full-refresh run of the affected models.

Recovery Time Objective (RTO): Medium (Minutes/Hours). Depends on how long the `dim\_customer` table takes to rebuild.

Risk: Temporary data inconsistency until the rebuild completes.

Final Senior Engineer Recommendation

Do not perform a Hard Drop.

I strongly advise moving the column to a `raw\_customers\_archive` table first, or using a View layer to hide the column from

downstream users while keeping the physical data intact for 30 days. This turns a "Catastrophic Recovery" (Scenario A) into a "Trivial Fix" (Scenario B).

Engineering counterpart
4:05 AM - Responded in 0s

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

Agent response

4:06 AM - Responded in 19s

To move this from "Blocking" to "Shipped," we need to shift from a destructive approach to a migratory approach.

Below is the Execution Roadmap. I have restructured this as a three-sprint rollout to ensure zero downtime and zero data loss.

Project: `legacy\_account\_id` Deprecation

Goal: Transition all dependencies to `crm\_account\_uuid` and safely remove legacy identifiers.

Phase 1: Identity Alignment (Sprint 1)

Focus: Eliminate the 5.8% data gap and unify the key.

| Step | Action | Owner | Timing | Deliverable |

| :--- | :--- | :--- | :--- | :--- |

| 1.1 | Data Backfill: Identify the 1,204 missing UUIDs; coordinate with CRM team to populate them in `raw.customers`. | Data Eng / CRM Admin | Week 1 | $\text{UUID Populated} = 100\%$ |

| 1.2 | Unified Key Logic: Update `int\_customer\_identity` to create `unified\_id = COALESCE(crm\_account\_uuid, legacy\_account\_id)`. | Data Eng | Week 1 | Updated Intermediate Model |

| 1.3 | Contract Update: Update Salesforce export logic to use `crm\_account\_uuid` as the matching key. | Analytics Eng | Week 2 | Verified SFDC Sync |

Phase 2: Downstream Migration (Sprint 2)

Focus: Moving the "Gravity" of the warehouse to the new key.

| Step | Action | Owner | Timing | Deliverable |

| :--- | :--- | :--- | :--- | :--- |

| 2.1 | SCD2 Refactor: Update `dim\_customer` merge logic to use `unified\_id` instead of `legacy\_account\_id`. | Data Eng | Week 3 | Validated `dim\_customer` |

| 2.2 | Mart Update: Update `fct\_subscriptions` and `churn\_training\_features` to join on `unified\_id`. | Analytics Eng | Week 3 | Zero-null join report |

| 2.3 | Shadow Testing: Run both old and new logic in parallel for 7 days; compare output totals. | QA / Analyst | Week 4 | Variance Report ($\$0 \Delta$) |

Phase 3: Safe Deprecation (Sprint 3)

Focus: Removing the column without risk of permanent loss.

| Step | Action | Owner | Timing | Deliverable |

| :--- | :--- | :--- | :--- | :--- |

| 3.1 | Soft Drop: Rename `legacy\_account\_id` $\rightarrow$ `deprecated\_legacy\_id` or hide via a View layer. | Data Eng | Week 5 | Broken query logs (Monitor) |

| 3.2 | Observation: Monitor query logs for 14 days to ensure no "hidden" dependencies remain. | Data Eng | Week 6-7 | Clean Audit Log |

| 3.3 | Hard Drop: Execute `DROP COLUMN` on `raw.customers` after final backup. | Data Eng | Week 8 | Final PR Closed |

Summary of Ownership & Accountability

Data Engineering (Lead): Responsible for the pipeline integrity, the SCD2 merge logic, and the final physical drop.

Analytics Engineering: Responsible for updating the marts (`fct\_subscriptions`, `dim\_customer`) and verifying the SFDC export.

CRM Admin: Responsible for the

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 8s

If I can only have one person double-check one specific thing, it is the SCD2 Merge Logic in ``dim_customer``.

Everything else in this plan is a "broken pipe" problem-if a join fails or an export breaks, the data is still there, and you just fix the code. But a mistake in an SCD2 merge is a "corrupted foundation" problem.

Why this is the critical point:

In an SCD2 table, the natural key is the only thing that tells the database: "This is the same person, just with a new address" vs. "This is a brand new person."

If the logic for switching from ``legacy_account_id`` to ``crm_account_uuid`` is slightly off (e.g., handling of NULLs, trimming of strings, or case sensitivity), the merge will fail to find a match for existing customers. Instead of updating the current record and expiring the old one, the system will simply insert a new record.

The result is a "silent disaster":

- Your customer count will suddenly double or triple.

- Your historical snapshots become fragmented.

- You cannot simply "re-run" the pipeline to fix it because you've now polluted the history of the table. Fixing this requires a painful, manual reconstruction of the dimension table from raw logs.

What I would ask the second pair of eyes to verify:

I would hand them the SQL for the ``MERGE`` statement and ask:

> "If ``crm_account_uuid`` is present but the ``legacy_account_id`` was the previous anchor, does this logic definitively map the new record to the existing history, or does it create a duplicate? Walk me through the join predicate for a record that is transitioning from legacy to UUID today."

Engineering counterpart

4:06 AM - Responded in 0s

Thanks - impact read and migration risks are clear. Migrate unmigrated accounts before dropping the legacy column in prod. Review complete - no further response needed.