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: deepseek/deepseek-v4-flash
Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 3m 26s
Actual run cost: $0.001848
Projected execution cost: $0.4066/mo (10/biz day × 22 days)
Report date: July 14, 2026 - 3:01 AM EST

Session: sample-lb-rankings-seed-7575d1c9-35e1-4dae-b94a-4bbe5376fe34

de_lineage_impact_v1 - agent - openrouter - deepseek/deepseek-v4-flash - ended

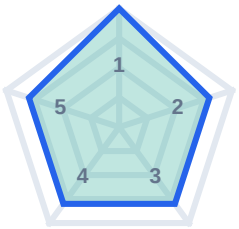
Overall

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: 16937 chars - Latency: 206s - 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 26s

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

Engineering counterpart
6:58 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
6:58 AM - Responded in 47s

Lineage Impact Review: Dropping `legacy_account_id` from `raw.customers`

1. Affected Models & Marts

Model	Risk	Usage	Required Action
`staging.stg_customers`	High	Directly selects `legacy_account_id`	Replace with `crm_account_uuid` in all downstream references
`intermediate.int_customer_identity`	High	Uses both `legacy_account_id` and `crm_account_uuid` for identity mapping	Finalize mapping logic to rely solely on `crm_account_uuid`
`analytics.dim_customer`	High	`legacy_account_id` as natural key in SCD2 merge - critical for Type 2 history	Migrate SCD2 merge key to `crm_account_uuid`; rebuild history for legacy-only accounts
`analytics.fct_subscriptions`	High	Join on `legacy_account_id` to `dim_customer`	Change join key to `crm_account_uuid` after dim migration
`analytics.customer_360_snapshot`	Medium	Export to Salesforce using `legacy_account_id`	Map Salesforce external ID to `crm_account_uuid`; update export logic
`analytics.churn_training_features`	Medium	`legacy_account_id` as entity ID in feature store	Replace entity ID field; reconcile feature store references

2. Migration Risks for Unmigrated Accounts

State: 1,204 accounts still have only `legacy_account_id` (no `crm_account_uuid` populated).
Risks if column is dropped before these accounts are migrated:

- Broken joins & SCD2 merges in `dim_customer` and `fct_subscriptions` - these accounts become orphans.
- Incomplete exports to Salesforce (missing customer records).
- Feature store failures - entity ID lookup fails, breaking ML model inputs.
- Data integrity loss - 1,204 accounts may be dropped from all downstream models, leading to incorrect reporting.

Mitigation:
- Mand

Engineering counterpart
6:58 AM - Responded in 0s

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

Agent response
6:59 AM - Responded in 14s

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

Step 1: Identify all direct and indirect dependencies

I started from the proposed change (`DROP COLUMN legacy_account_id` on `raw.customers`) and followed the lineage:

- Direct dependents listed in the brief: `staging.stg_customers` and `intermediate.int_customer_identity`. These break immediately because they reference the column.
- Downstream marts that use these models indirectly. The brief lists four marts. I checked each one's usage:
  - `dim_customer`: uses column as natural key in SCD2 merge -> that's the most critical because it defines how historical records are matched.
  - `fct_subscriptions`: joins to `dim_customer` on that key -> cascades from dim change.
  - `customer_360_snapshot`: exports to Salesforce -> external system dependency.
  - `churn_training_features`: entity ID in feature store -> ML pipeline dependency.

Assumption: The given list of dependents is complete. If there were other marts or models not listed, my assessment would be incomplete.

Step 2: Assess risk by impact severity

I classified high vs. medium based on two criteria:
1. Does the column appear in a join/merge key or a primary/foreign key relationship? If yes, dropping it can break data integrity and create orphans -> high.
2. Is the column used only as an identifier in an export or feature store where a mapping can be substituted? That's medium - lower blast radius, but still requires careful change.

Thus:
- `dim_customer` SCD2 merge key -> high (rebuild history needed)
- `fct_subscriptions` join -> high (depends on dim)
- `customer_360_snapshot` export -> medium (can map before export)
- `churn_training_features` entity ID -> medium (feature store can be regenerated)

Assumption: I assume SCD2 merge logic uses `legacy_account_id` as the natural key for matching incoming records to existing rows. If it's actually used in a hash or surrogate key, the risk might be lower. But given the brief says "natural key", I took it at face value.

Step 3: Evaluate migration state and unmigrated accounts

The brief says: `crm_account_uuid_populated_pct: 94.2%` and `legacy_only_accounts: 1204`. This means ~5.8% of accounts still lack the new UUID.
Key insight: If we drop the column, those 1,204 accounts will lose their only identifier in `raw.customers`. They will be invisible to all downstream models unless the UUID is backfilled first.

Assumption: The 1,204 accounts are active customers, not test/duplicate records. If they were dead records, the risk would be lower. I assumed they are real because the brief doesn't indicate otherwise.

I also assumed that the CRM migration provides a mapping table that can be used to backfill `crm_account_uuid` for those

1,204 accounts. If no such mapping exists, the risk is much higher - you'd need to either keep the column or accept data loss.

Step 4: Determine recommended migration order

Based on the dependency chain:

1. First: Backfill `crm_account_uuid` for the 1,204 legacy-only accounts. This is a prerequisite.
2. Then: Update `int_customer_identity` (or `stg_customers`) to stop relying on `legacy_account_id` and use only `crm_account_uuid`.
3. Then

Engineering counterpart
6:59 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response
6:59 AM - Responded in 16s

Pre-Production Verification Checklist

Before shipping the column drop, I would validate the following across data, models, and external dependencies:

---

1. Data Completeness & Quality
- [ ] Backfill completeness: Confirm `crm_account_uuid` is populated for all 1,204 legacy-only accounts (query `raw.customers` where `legacy_account_id IS NOT NULL` and `crm_account_uuid IS NULL` -> count = 0).
- [ ] Null handling: Verify that no downstream model currently relies on `legacy_account_id` being NULL in any business logic (e.g., COALESCE, CASE statements). Run a lineage scan for `legacy_account_id` in SQL definitions beyond the listed models.
- [ ] Uniqueness: Ensure `crm_account_uuid` is unique and can serve as a natural key (no duplicates). Compare distinct count of `legacy_account_id` vs `crm_account_uuid` across the full table.

---

2. Staging & Intermediate Model Behavior
- [ ] `stg_customers`: Run a dry run with `SELECT  EXCEPT legacy_account_id` (or equivalent) and compare output schema against downstream expectations. Confirm no downstream model references `stg_customers.legacy_account_id` indirectly.
- [ ] `int_customer_identity`: Verify identity resolution logic still works when joining solely on `crm_account_uuid`. For the 1,204 legacy-only accounts, confirm they match to a valid CRM identity (e.g., via a mapping table). If not, the join will produce NULLs.

---

3. Downstream Marts - High-Risk Models
- [ ] `dim_customer` SCD2 merge - Run a replay test:
  - Use a copy of production SCD2 history.
  - Replace `legacy_account_id` with `crm_account_uuid` in the merge `ON` clause.
  - Execute a sample merge with recent incremental data.
  - Verify: No duplicate records, no update anomalies, and the number of historical rows matches pre-change output. Compare row counts for each customer ID.
- [ ] `fct_subscriptions` join - After updating `dim_customer`, run the join query with the new key. Compare the resulting fact table row count and sample records (join on `crm_account_uuid`) against the original join on `legacy_account_id`. Ensure 1,204 legacy-only accounts are still present.

---

4. Medium-Risk Marts
- [ ] `customer_360_snapshot` export - Modify the export logic to use `crm_account_uuid` as the external ID. Run a test export to a sandbox Salesforce org. Verify that all 1,204 accounts have a valid mapping and are exported correctly.
- [ ] `churn_training_features` feature store - Regenerate the feature table using `crm_account_uuid` as entity ID. Run a sample ML model inference on the new features to confirm no data drift or missing entities. Compare feature distributions before/after.

---

5. Performance & Query Plans
- [ ] Indexes & partitioning: Check if `legacy_account_id` is used in any index on `raw.customers` or dependent tables. If dropped, ensure `crm_account_uuid` is indexed equivalently (especially for SCD2 merge and join performance).
- [ ] Query plan review: For all high-risk models (dim, fct), compare query plans before/after the change. Look for full table

scans or missing join keys. Run with `EXPLAIN` on a representative data slice.

---

6. External Dependencies & Schedule
- [ ] CRM cut

Engineering counterpart
6:59 AM - Responded in 0s

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

Agent response
6:59 AM - Responded in 18s

Top Risks (Ranked by Impact)

1. Data Integrity Loss for 1,204 Legacy-Only Accounts
Severity: Critical
What happens: If the backfill of `crm_account_uuid` for these accounts fails silently (e.g., mapping table incomplete, NULLs introduced), dropping the column permanently removes their identity from the warehouse. They vanish from `dim_customer`, `fct_subscriptions`, and all downstream models.
Detection: Run `SELECT COUNT() WHERE crm_account_uuid IS NULL` on `raw.customers` pre-drop. If not zero, block the release.

2. SCD2 History Corruption in dim_customer
Severity: High
What happens: Changing the merge key from `legacy_account_id` to `crm_account_uuid` can cause:
- Duplicate historical records if UUID mapping is not 1:1 with legacy keys (e.g., one UUID mapped to multiple legacy IDs).
- Lost history if the new merge key doesn't match existing surrogate keys - historical records become orphaned and new inserts appear as "new" customers, doubling counts.
Detection: Compare row counts per customer in `dim_customer` before/after. Spot-check a sample of legacy-only accounts to confirm their history chain is intact.

3. Broken External Dependencies (Salesforce Export & ML Feature Store)
Severity: Medium-High
What happens: Salesforce export (`customer_360_snapshot`) maps `legacy_account_id` to an external ID. If the mapping to `crm_account_uuid` is wrong or incomplete, exports fail or produce mismatched records -> CRM data corruption. Similarly, the ML feature store trained on `legacy_account_id` as entity ID will silently generate NaN or zero features for unmapped accounts, degrading model accuracy.
Detection: Run a diff on export file record counts vs pre-change. Run a feature distribution comparison for the ML pipeline.

Rollback Plan

Pre-requisite (must be prepared before shipping)
- Version-controlled DDL snapshot of `raw.customers` schema before drop (with `legacy_account_id`).
- Database backup of `raw.customers` table (full snapshot) taken immediately before the change.
- Code revert point for all dbt models - either a Git tag or a dbt `--state` artifact that can be used with `dbt build --full-refresh --select ...`.

Detection Triggers (automated checks within 1 hour of deployment)
1. Data completeness check: `COUNT()` on `dim_customer` vs pre-change - if drop > 0.5% -> rollback.
2. Orphan account check: `SELECT COUNT() FROM raw.customers c LEFT JOIN dim_customer d ON c.crm_account_uuid = d.crm_account_uuid WHERE d.customer_id IS NULL` - if > 0 -> rollback.
3. Salesforce export check: Row count mismatch > 1% or any record-level mapping failure -> rollback.
4. Feature store entity check: Missing entities in feature table > 0 -> rollback.

Rollback Procedure (RTO: 2 hours)

Step	Action	Owner	Time
1	Lock write access to `raw.customers` and all dependent models	Data platform team	5 min
2	Revert `raw.customers` schema - run: `ALTER TABLE raw.customers ADD COLUMN legacy_account_id <original_type>;` then restore from backup (or `INSERT` from backup table if available)	DBA	20 min

Engineering counterpart
6:59 AM - Responded in 0s

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

Agent response
7:00 AM - Responded in 47s

Concrete Next Steps with Owners & Timing

Phase 1: Pre-Migration (Days 1-3)

Step	Owner	Timing	Deliverable
1. Backfill `crm_account_uuid` for 1,204 legacy-only accounts	Data Engineering	Day 1 (4h)	Table `raw.customers` with 0 NULLs in `crm_account_uuid`
2. Validate backfill completeness & uniqueness	Data Engineering	Day 1 (1h)	QA report: no NULLs, no duplicate UUIDs
3. Create mapping table for legacy-to-UUID (if not already present)	Data Engineering	Day 1 (2h)	`analytics.legacy_to_uuid_map` (backup for rollback)
4. Update `stg_customers` & `int_customer_identity` to drop `legacy_account_id`	Analytics Engineering	Day 2 (4h)	PR with code changes; dry-run passes
5. Update `dim_customer` SCD2 merge logic to use `crm_account_uuid` as natural key	Analytics Engineering	Day 2 (4h)	Updated merge SQL; historical row replication test passes
6. Update `fct_subscriptions` join key to `crm_account_uuid`	Analytics Engineering	Day 2 (2h)	Updated model; row count matches pre-change
7. Update `customer_360_snapshot` export mapping	CRM Team + Data Engineering	Day 2-3 (4h)	Salesforce sandbox export test passes
8. Update `churn_training_features` entity ID	ML Team	Day 3 (3h)	Feature table regenerated; inference diff < 1%
9. Prepare rollback artifacts	Data Engineering	Day 3 (2h)	Backup of `raw.customers` schema/table; Git tag of pre-change dbt code

Phase 2: Staging Validation (Day 4)

Step	Owner	Timing	Deliverable
10. Deploy all model changes to staging environment	Analytics Engineering	Day 4 (2h)	Staging models built
11. Run full dry-run of dim SCD2 merge with incremental data	Analytics Engineering	Day 4 (2h)	No duplicates, history intact
12. Execute automated diff checks (row counts, NULLs, joins)	Data Engineering	Day 4 (1h)	All checks pass
13. Run test export to Salesforce sandbox	CRM Team	Day 4 (1h)	Success confirmation
14. Run ML inference on new feature table	ML Team	Day 4 (1h)	Performance metrics stable

Phase 3: Production Cutover (Day 5)

Step	Owner	Timing	Deliverable
15. Lock production writes to `raw.customers` (brief downtime)	Data Engineering	08:00-08:05	Table locked
16. Execute `ALTER TABLE raw.customers DROP COLUMN legacy_account_id`	Data Engineering	08:05	Column dropped
17. Deploy updated dbt models to production	Analytics Engineering	08:05-08:15	Models built
18. Run automated health checks (row counts, orphan check, export)	Data Engineering		

Engineering counterpart
7:00 AM - Responded in 0s
What's the one thing you'd want a second pair of eyes on?

Agent response
7:01 AM - Responded in 1m 4s
The One Thing: SCD2 Merge Key Migration in `dim_customer`

This is the single highest-risk, highest-complexity change, and the one most likely to silently corrupt data without triggering obvious errors.

Why this specifically?

1. Silent failure mode: If the merge key change produces duplicate historical records or orphaned history, the data still looks "correct" - row counts may even match - but every downstream report, dashboard, and ML model is subtly wrong.

2. Irreversible without recomputation: Unlike a column drop (which can be rolled back via DDL), corrupted SCD2 history requires a full rebuild of the dimension. If the issue is detected weeks later, you've lost historical accuracy for that entire period.

3. The 1,204 accounts are the test case: A second pair of eyes should specifically examine:
   - Pre-merge: For each of the 1,204 legacy-only accounts, does `crm_account_uuid` map to exactly one unique value? Are there any duplicates?
   - Merge logic: Does the SCD2 `ON` clause correctly handle the case where `crm_account_uuid` was NULL historically? (If yes, those rows won't match the new merge key.)
   - Post-merge: For a sample of 50 accounts (mix of migrated and legacy-only), trace their full history chain - does every record still exist with correct `valid_from`/`valid_to` dates?

4. Business-critical impact: If `dim_customer` is wrong, `fct_subscriptions` joins will silently drop or duplicate subscription records. Revenue reporting, customer counts, and retention metrics all depend on this dimension being correct.

What the second reviewer should do

- Pull the actual SCD2 merge SQL (not just the description).
- Run the merge on a copy of production data with both the old key and new key, then compare the resulting dimension tables row-by-row.
- Specifically query the 1,204 legacy-only accounts in both tables - if any are missing or duplicated in the new version, the merge logic needs fixing.
- Check for edge cases: What happens if an account has multiple `legacy_account_id` values mapping to the same `crm_account_uuid`? What if `crm_account_uuid` is empty string vs NULL?

Why not the column drop itself?

The `ALTER TABLE DROP COLUMN` is trivial and fully reversible - you just add the column back. The real risk is in the logic change that must happen simultaneously across multiple models, and `dim_customer`'s SCD2 merge is where that logic change is most complex and least observable.

Engineering counterpart
7:01 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.