Vantage RuntimeAI

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

8.0/10

Overall score

Run summary

Scenario: Schema Drift Review
Model: openai/gpt-4.1-nano
Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 39s
Actual run cost: $0.001160
Projected execution cost: $0.7656/mo (30/biz day × 22 days)
Report date: July 14, 2026 - 3:19 AM EST

Session: sample-lb-rankings-seed-631f51ee-05fc-43aa-adb7-3bf416bc3de0

de_schema_drift_v1 - agent - openrouter - openai/gpt-4.1-nano - ended

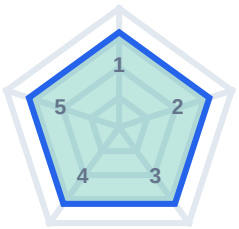
Overall

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	Breaking change detection	8/10
2	References both schemas	8/10
3	Migration risk awareness	8/10
4	Structured diff summary	8/10
5	Actionable next steps	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.

- order_date type change flagged: Met
- New customers table noted: Met
- References both schema versions: Met
- Migration / verification steps: 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: 4 - Counterpart msgs: 5 - Output: 12366 chars - Latency: 39s - Criteria: 4/4

Score analysis

Flags order_date type change (DATE -> TIMESTAMP_NTZ) as a breaking concern.

Score snapshot

Solid: Solid technical task response with a few gaps.

Interpretation

Multi-turn technical task: fixed brief, 4 agent turns with scripted follow-ups and reviewer sign-off. 4 of 4 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

+ Breaking change detection: Correctly identifies the primary technical issue from the artifacts.
+ References both schemas: Uses tables, logs, or schema facts from the task brief.
+ Migration risk awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
+ Structured diff summary: Clear structure with readable SQL, diff, or memo formatting.
+ Actionable next steps: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Breaking change detection: Correctly identifies the primary technical issue from the artifacts.
- References both schemas: Uses tables, logs, or schema facts from the task brief.
- Migration risk awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- Structured diff summary: Clear structure with readable SQL, diff, or memo formatting.
- Actionable next steps: 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

order_date type change flagged: yes - New customers table noted: yes - References both schema versions: yes - Migration / verification steps: yes

CAVEATS

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

Transcript

Session

Scenario: Schema Drift Review
Agent response <-> Engineering counterpart
Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 39s

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

Engineering counterpart
7:18 AM

Task brief - Schema drift review (moderate complexity)

Compare schema v1 (Q1) vs v2 (Q2) and summarize breaking changes.

Schema v1 (2024-Q1):
```json
{
  "version": "2024-Q1",
  "tables": {
    "orders": {
      "columns": {
        "order_id": "NUMBER",
        "customer_id": "NUMBER",
        "order_date": "DATE",
        "total_usd": "NUMBER(12,2)",
        "status": "VARCHAR(32)"
      }
    },
    "order_items": {
      "columns": {
        "order_id": "NUMBER",
        "sku": "VARCHAR(64)",
        "qty": "NUMBER",
        "unit_price": "NUMBER(10,2)"
      }
    }
  }
}
```

Schema v2 (2024-Q2):
```json
{
  "version": "2024-Q2",
  "tables": {
    "orders": {
      "columns": {
        "order_id": "NUMBER",
        "customer_id": "NUMBER",
        "order_date": "TIMESTAMP_NTZ",
        "total_usd": "NUMBER(12,2)",
        "status": "VARCHAR(32)",
        "region_code": "VARCHAR(8)"
      }
    },
    "order_items": {
      "columns": {
        "order_item_id": "NUMBER",
        "order_id": "NUMBER",
        "sku": "VARCHAR(64)",
        "qty": "NUMBER",
        "unit_price": "NUMBER(10,2)"
      }
```

```
    },
    "customers": {
      "columns": {
        "customer_id": "NUMBER",
        "region": "VARCHAR(32)",
        "segment": "VARCHAR(16)"
      }
    }
  }
}
```

Agent response
7:19 AM - Responded in 8s

Breaking Changes Between v1 (2024-Q1) and v2 (2024-Q2):

1. Data Type Changes
- `orders.order_date`:
  - v1: `DATE`
  - v2: `TIMESTAMP_NTZ`
  - Impact: Consumers expecting only date values may encounter issues if time components are used or if strict type checks are implemented. This change is potentially breaking if consumers are not prepared for timestamp data.

2. Removed Columns
- `order_items.sku`:
  - v1: exists
  - v2: removed
  - Note: The `sku` column is missing in v2's `order_items`. This is a breaking change as clients expecting this data will find it missing.

3. Changed Column Name
- `order_items` table:
  - v1: `qty`
  - v2: renamed to `order_item_id` (which replaces `qty`'s previous position)
  - Impact: The original `qty` field is missing; instead, a new `order_item_id` is introduced. Consumers relying on `qty` will break unless adapted.

4. Newly Added Columns (Optional for Compatibility)
- `orders.region_code`:
  - v2: added
  - Impact: Non-breaking if clients ignore the new column, but may cause issues if they expect a fixed schema or attempt to write to this column.

- `customers` table added in v2:
  - Impact: This is a new table, which does not affect existing data consumers unless they rely on the entire schema.

5. New Table
- `customers` table:
  - v2: added
  - Impact: Completely new and non-breaking for existing schemas unless clients attempt to access this non-existent table.

Summary of Breaking Changes:
- The change of `orders.order_date` from `DATE` to `TIMESTAMP_NTZ`.
- Removal of the `order_items.sku` column.
- Renaming or replacing `order_items.qty` with `order_item_id`.
- Potential missing data if clients expect the `sku` column.
- Any client code that assumes the presence of `qty` in `order_items` will break.

Non-Breaking Additions:
- `orders.region_code` column
- New `customers` table with `customer_id`, `region`, and `segment`.

Migration Risks and Verification Steps:

Risks:
- Data consumers expecting consistent data types (e.g., `order_date`) may face errors or unexpected behavior.
- Loss of `sku` data in `order_items` may cause downstream processing failures.
- Schema mismatch in clients that reference renamed or removed columns.
- Incomplete handling of new columns and tables may lead to data integrity or reporting gaps.

Verification Steps:
1. Type Compatibility Checks:
   - Validate that consumers of `orders.order_date` can handle `TIMESTAMP_NTZ` instead of `DATE`.

2. Schema Validation:
   - Confirm that all clients are updated to handle the missing `sku` in `order_items` or modify data extraction accordingly.
   - Check for code dependencies on `order_items.qty`; adjust for `order_item_id` if necessary.

3. Data Completeness:
   - Assess whether `sku` data is being migrated or if its removal in v2 is intentional.

4. Testing:
   - Run schema validation and integration tests ensuring all dependent applications handle schema changes gracefully.

5. Documentation and Communication:
   - Document schema changes clearly.
   - Notify downstream teams of the change in data types and schema structure.

Summary:
The most critical

Engineering counterpart
7:19 AM - Responded in 0s

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

Agent response
7:19 AM - Responded in 10s

Certainly! Here's a detailed walkthrough of my reasoning process, including the assumptions made:

Step 1: Schema Comparison Overview
Objective: Identify what has changed from v1 to v2 - focusing on breaking vs. non-breaking changes.

Approach:
- Review each table and its columns in both schemas.
- Note differences in presence, data types, names, and structure.

Step 2: Identify Explicit Structural and Data Type Changes

Assumption:
- Changes in data types (like from `DATE` to `TIMESTAMP_NTZ`) can be breaking if existing consumers expect a specific data format.
- Removing or renaming columns is inherently breaking, unless the data consumers are updated or tolerant.
- Addition of new columns or tables is non-breaking unless consumers have strict schema expectations.

Findings:
- `orders.order_date`: changed from `DATE` (v1) to `TIMESTAMP_NTZ` (v2)-potentially breaking due to data type change.

- `order_items.sku` exists in v1 but absent in v2-removal, which is breaking.
- `order_items.qty` in v1 seems replaced by `order_item_id` in v2-renaming or redefinition, which breaks consumers expecting the old column (or missing it).
- `orders.region_code`: new column; non-breaking addition.
- `customers` table added: non-breaking new table.

Step 3: Detect Removed Columns
- The absence of `sku` in `order_items` in v2 implies data loss for that attribute, which is a breaking change.
- The `qty` column is missing in v2 - either replaced or renamed to `order_item_id`.
  - Assumption: Since `qty` is no longer present, clients expecting it will break unless modified.

Step 4: Detect Renamed or New Columns
- The appearance of `order_item_id` in v2 could indicate a new primary key or identifier replacing `sku` or `qty`.
- I assume the schema change reflects a design update, so clients expecting `qty` will need adjustment.

Step 5: Examine New Tables and Columns
- `orders.region_code` is an optional, non-breaking addition.
- `customers` table is entirely new, so it won't affect existing consumers unless they query for it.

Step 6: Summarize Breaking vs. Non-Breaking
- I classify changes affecting existing data consumers:
  - Data type change in `order_date` (potentially breaking).
  - Removal of `sku`.
  - Renaming/replacement of `qty`.
- Additions that are non-breaking: new columns, new tables.

Step 7: Consider Assumptions Made
- Assumption 1: All consumers expect the columns to be present unless explicitly handled.
- Assumption 2: Data type changes from `DATE` to `TIMESTAMP_NTZ` are potentially breaking unless consumers are flexible.
- Assumption 3: The removal of `sku` is a breaking change because downstream systems may rely on SKU data.
- Assumption 4: Renaming `qty` to `order_item_id` means clients expecting `qty` will break unless updated.
- Assumption 5: New columns or tables do not affect existing consumers unless they explicitly query those.

Step 8: Verification and Risk Assessment
- I recommend that consumers validate their compatibility with the new schema, especially for `order_date` data type and the missing `sku`.
- Ensure data migration or handling of missing columns is coordinated.
- Plan schema validation tests and code updates accordingly.

Summary of the Reasoning:
- Comparison of explicitly provided schema snapshots.
- Identification of differences based on presence, data types, and naming.
- Application

Engineering counterpart
7:19 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response
7:19 AM - Responded in 9s

Before deploying this schema change to production, I would verify the following to mitigate risks and ensure a smooth transition:

---

1. Compatibility of Data Types
- Verify consumers' tolerance for `order_date` type change:
  - Confirm whether client applications, ETL pipelines, and BI tools can handle `TIMESTAMP_NTZ` instead of `DATE`.
  - Ensure that date-only logic (e.g., date truncations, date filters) behave correctly with the timestamp data.

2. Data Migration and Integrity
- Assess impact of removing `sku` from `order_items`:
  - Confirm if existing data or downstream processes depend on `sku`.
  - Plan for data migration or handling of missing data if `sku` is still needed.

3. Schema Alignment for Clients
- Update client schemas or code that depend on the previous schema:
  - For the renamed or replaced `qty` column, ensure client applications are updated to use `order_item_id`, or handle the absence of `qty`.
  - Communicate the change clearly to all consumers and downstream teams.

4. Testing
- Perform comprehensive testing:
  - Schema validation tests to confirm the correct schema is deployed.
  - Integration tests with downstream systems to check data ingestion, processing, and reporting.
  - Data validation checks to ensure data consistency and completeness after migration.

5. Data Validation
- Verify data correctness:
  - Ensure that existing date data complies with the new timestamp format or handle old data accordingly.
  - Confirm that records with `sku` are handled properly, or that the missing `sku` won't cause issues downstream.

6. Monitoring and Rollback Plan
- Establish monitoring for issues post-deployment:
  - Track error rates, failed queries, or mismatched data.
- Prepare a rollback plan in case any breaking issues are detected, especially related to data type incompatibilities or missing columns.

7. Documentation and Communication
- Document the schema change comprehensively:
  - Include details on data type changes, removed columns, new columns, and tables.
- Notify all relevant teams, including data consumers, engineers, and business stakeholders, about the upcoming change, expected impacts, and required updates.

8. Coordinate Data Migration
- If historical data or old schemas need to be adjusted, plan for data migration or transformation scripts before switch-over.

Summary:
- Pre-deployment, I'd verify data type compatibility, data integrity, client update readiness, and conduct thorough testing.
- Communication and monitoring are essential for quickly identifying and resolving any issues after deployment.

Would you like a checklist or detailed plan for each step?

Engineering counterpart
7:19 AM - Responded in 0s

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

Agent response
7:19 AM - Responded in 12s

Top Risks:

1. Data Type Compatibility Issues:
   - Changing `orders.order_date` from `DATE` to `TIMESTAMP_NTZ` may cause failures if consuming systems or queries expect date-only values.

2. Missing or Replaced Columns:
   - Removal of `order_items.sku` could break downstream processes relying on SKU data.
   - Replacement of `qty` with `order_item_id` in `order_items` may cause schema mismatches and failures in data processing or reporting.

3. Client Application Breakage:
   - Existing code expecting `qty` could fail or produce errors if `qty` is missing.
   - Any applications that do not handle timestamps properly could generate errors or inaccurate data.

4. Data Migration Risks:
   - If historical data needs to be transformed or migrated (e.g., converting `DATE` to `TIMESTAMP_NTZ`), errors or data loss could occur.

5. Unanticipated Dependencies or Assumptions:
   - Downstream systems, dashboards, or reports might rely on the removed or renamed columns, causing failures or inconsistent outputs.

Rollback Plan:

1. Immediate Halt and Issue Diagnosis
- As soon as a problem is detected, cease further deployments.
- Identify the root cause (e.g., schema mismatch, data type incompatibility).

2. Revert Schema Changes
- Restore the previous schema version (`2024-Q1`) with:
  - Original data types for `order_date` (`DATE`)
  - All columns present, including `sku` and `qty`
  - No new columns (e.g., `region_code`) or tables (`customers`)

3. Data Handling
- If any data was migrated or transformed based on the new schema, revert or re-migrate the data to prevent inconsistencies.
- Validate data integrity.

4. Notify Stakeholders
- Communicate the rollback to all relevant teams.
- Inform them of the failure reason and next steps.

5. Validate and Re-test
- Ensure the restored schema and data are consistent.
- Conduct targeted tests to verify stability.

6. Post-Rollback Actions
- Analyze the failure causes.
- Adjust the schema migration plan, including:

- Additional testing
- More gradual rollout (e.g., feature flag or phased migration)
- Enhanced validation checks

7. Reschedule Deployment
- Once issues are resolved, plan a controlled, monitored re-deployment, possibly with mitigations for identified risks.

Summary:
Risks:
- Compatibility issues with data types
- Missing or mismatched columns leading to client failures
- Data integrity and migration errors

Rollback plan:
- Immediate schema and data reversion to the previous stable version
- Clear communication and validation
- Post-incident review and adjustment before re-deploying

Would you like a detailed checklist for this rollback process or specific mitigation strategies?

Engineering counterpart
7:19 AM - Responded in 0s

Thanks - useful breakdown of breaking vs additive changes. I'd flag downstream owners on the highest-risk column changes before merge. Review complete - no further response needed.