



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: meta-llama/llama-3.3-70b-instruct

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

Duration: 3m 8s

Actual run cost: \$0.001340

Projected execution cost: \$0.8844/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 3:19 AM EST

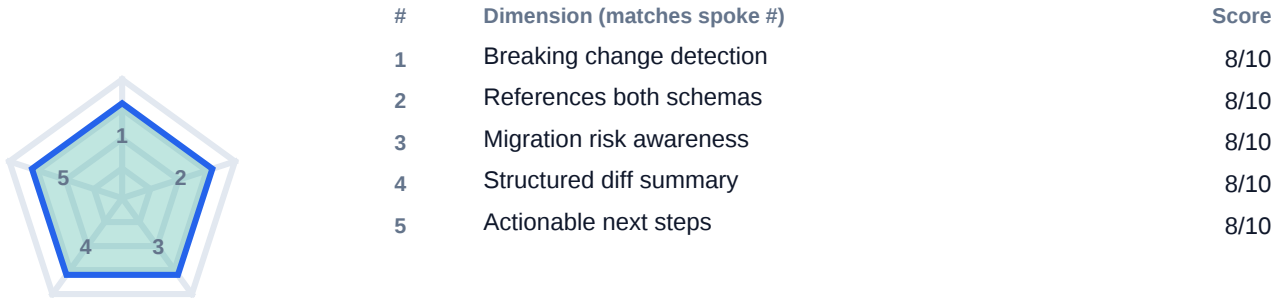
Overall

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.

- 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: 12719 chars - Latency: 188s - 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: 3m 8s

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

Engineering counterpart

7:16 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

7:17 AM - Responded in 40s

Breaking Changes

The following changes are considered breaking as they may affect existing queries or applications:

1. Data Type Change: In the `orders` table, the `order\_date` column's data type has changed from `DATE` to `TIMESTAMP\_NTZ`. This change may impact date-based queries or applications that rely on the specific date format.
2. New Required Field: In the `orders` table, a new column `region\_code` of type `VARCHAR(8)` has been added. Although the schema does not explicitly mark it as required, its presence may be mandatory for new records, potentially breaking existing insert or update queries that do not provide a value for this column.
3. Column Rename: In the `order\_items` table, the column `order\_id` remains, but a new column `order\_item\_id` of type `NUMBER` has been added. This suggests that each order item now has a unique identifier, which might require updates to queries or applications that previously relied solely on the `order\_id` for identifying order items.

Non-Breaking Additions

The following additions are considered non-breaking as they do not alter existing structures but rather add new ones:

1. New Table: A new table named `customers` has been added. This table includes columns for `customer\_id`, `region`, and `segment`, providing additional customer information.
2. New Column in Existing Table: The `order\_item\_id` column in the `order\_items` table is a new addition. This column provides a unique identifier for each order item, which can enhance data management and queries without disrupting existing structures.

Migration Risks and Recommended Verification Steps

To mitigate risks associated with the schema migration, consider the following verification steps:

1. Test Date-Based Queries: Verify that all queries and applications using the `order\_date` column in the `orders` table are compatible with the new `TIMESTAMP\_NTZ` data type.
2. Update Insert/Update Queries: Modify insert and update queries for the `orders` table to include the new `region\_code` column, ensuring that a valid value is provided to avoid errors.
3. Validate Unique Identifiers: Confirm that the introduction of `order\_item\_id` in the `order\_items` table does not conflict with existing unique identifier logic. Test queries and applications to ensure they correctly handle this new identifier.
4. Integrate Customer Data: If applicable, integrate the new `customers` table into relevant queries and applications. Ensure that joins and data manipulation involving customer information are correctly implemented.
5. Backward Compatibility Testing: Perform thorough testing to ensure that existing queries, reports, and applications remain functional after the schema migration. Pay particular attention to areas affected by breaking changes.
6. Documentation Update: Update documentation to reflect the changes in the schema, including new tables, columns, and data types. This ensures that developers and users have accurate information for future development and troubleshooting.

Engineering counterpart

7:17 AM - Responded in 0s

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

Agent response

7:18 AM - Responded in 1m 5s

Here's a step-by-step breakdown of my reasoning and the assumptions I made:

Step 1: Identify the Schemas

I started by identifying the two schemas provided:

- Schema v1 (2024-Q1): The original schema with two tables, `orders` and `order\_items`.
- Schema v2 (2024-Q2): The updated schema with three tables, `orders`, `order\_items`, and a new table `customers`.

Step 2: Compare Tables

Next, I compared the tables between the two schemas:

- The `customers` table is new in Schema v2, which is a non-breaking addition since it does not alter existing tables.
- Both `orders` and `order\_items` tables exist in both schemas but have changes.

Step 3: Analyze Changes in the `orders` Table

In the `orders` table:

- Data Type Change: The `order\_date` column changes from `DATE` to `TIMESTAMP\_NTZ`. I assumed this could be a breaking change because it might affect queries that rely on the date format or applications that parse this field.
- New Column: The `region\_code` column is added. Although the schema doesn't specify it as required, I considered it a potential breaking change because existing insert or update queries might not provide a value for this column, which could lead to errors if the column is indeed mandatory.

Step 4: Analyze Changes in the `order\_items` Table

In the `order\_items` table:

- New Column: The `order\_item\_id` column is introduced. I saw this as a non-breaking addition because it adds new functionality (unique identifiers for order items) without removing or changing existing columns. However, I noted it could be considered a breaking change if applications or queries relied on the assumption that `order\_id` was unique for each item, which is no longer the case.

Step 5: Consider Migration Risks

When considering migration risks:

- I assumed that applications or queries using the `order\_date` field might need updates to handle the new data type correctly.
- The addition of `region\_code` in the `orders` table and `order\_item\_id` in the `order\_items` table could require updates to existing queries or applications to ensure compatibility.
- The introduction of the `customers` table, while non-breaking, could still require integration efforts to leverage the new customer data effectively.

Assumptions Made

1. Mandatory Fields: I assumed that new columns added to existing tables (like `region\_code`) might be mandatory, even though the schema doesn't explicitly state this. This assumption is based on typical database design practices where new columns are often required to ensure data consistency.
2. Query and Application Dependence: I assumed that existing queries and applications might heavily depend on the original schema structure, including data types and the absence of certain columns. This means that changes, even if theoretically non-breaking, could still cause issues if these dependencies are not properly managed.
3. Backward Compatibility: I considered the importance of ensuring that existing queries, reports, and applications remain functional after the schema migration, which involves thorough testing to validate backward compatibility.

By walking through these steps and making these assumptions, I identified potential breaking changes, non-breaking

additions, and migration risks associated with moving from Schema v1 to Schema v2.

Engineering counterpart
7:18 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

7:19 AM - Responded in 1m 2s

Before shipping the updated schema to production, I would verify the following:

Functional Verification

1. Query Compatibility:

Test all existing queries, including selects, inserts, updates, and deletes, to ensure they work as expected with the new schema.

Pay special attention to queries that use the `order\_date` field, given its data type change.

2. Application Integration:

Verify that all applications, services, and APIs that interact with the `orders` and `order\_items` tables can handle the changes, including the new `region\_code` and `order\_item\_id` columns.

Test any application-specific logic that relies on the original schema, such as date parsing or unique identifier generation.

3. Data Integrity:

Validate that data migration scripts (if any) correctly transform existing data to conform to the new schema.

Check for any data inconsistencies or errors that might arise from the schema changes.

Performance Verification

1. Query Performance:

Monitor query execution times and resource utilization to ensure that the schema changes do not introduce performance regressions.

Test queries with varying data volumes and complexities to simulate real-world scenarios.

2. Indexing and Optimization:

Review and update indexing strategies as needed to accommodate the changed schema.

Ensure that query optimization techniques, such as caching or materialized views, are still effective.

Security and Access Control Verification

1. Access Control:

Verify that access control mechanisms, such as grants, roles, and permissions, are updated to reflect the new schema.

Ensure that sensitive data, like customer information, is properly secured and accessible only to authorized users.

2. Data Encryption:

If data encryption is used, confirm that it is still effective and properly configured for the updated schema.

Operational Verification

1. Backup and Recovery:

Test backup and recovery processes to ensure they work correctly with the new schema.

Verify that backups are properly formatted and can be restored in case of data loss or corruption.

2. Monitoring and Logging:

Update monitoring and logging configurations to account for the changed schema.

Ensure that relevant metrics and logs are collected to facilitate troubleshooting and performance optimization.

Testing and Validation

1. Unit Testing:

Develop and run unit tests to validate individual components and functions that interact with the updated schema.

2. Integration Testing:

Perform integration testing to verify that the updated schema works correctly with other system components and dependencies.

3. End-to-End Testing:

Conduct end-to-end testing to simulate real-world scenarios and validate the entire system's functionality with the new schema.

By thoroughly verifying these aspects, you can ensure a smooth transition to the updated schema and minimize the risk of issues in production.

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 21s

Top Risks

Based on the schema changes and potential impact on the system, the top risks are:

1. Query Compatibility Issues: Existing queries may not work as expected due to the changes in the `order\_date` data type and the addition of new columns, leading to errors or incorrect results.
2. Data Integrity Problems: The data migration process may introduce errors or inconsistencies, particularly if the transformation scripts are not thoroughly tested or if there are issues with data validation.
3. Performance Degradation: The updated schema may lead to performance issues, such as slower query execution times or increased resource utilization, which could impact the overall system responsiveness and user experience.
4. Application Integration Failures: Applications or services that interact with the `orders` and `order\_items` tables may not be properly updated to handle the schema changes, resulting in errors or failures.

Rollback Plan

In case the updated schema does not work as expected, a rollback plan should be in place to quickly revert to the previous schema version. Here's a step-by-step rollback plan:

1. Assess the Issue: Identify the root cause of the problem and determine the scope of the impact.
2. Notify Stakeholders: Inform relevant teams, stakeholders, and users about the issue and the planned rollback.
3. Restore Previous Schema: Apply the previous schema version to the database, which may involve:
 - Reverting the database to a previous backup or snapshot.
 - Applying a reverse migration script to undo the changes.
 - Manually updating the schema to match the previous version.
4. Revert Data Changes: If data was migrated or transformed as part of the schema update, revert these changes to ensure data consistency. This may involve:
 - Applying a reverse data migration script.
 - Manually updating or correcting data to match the previous schema.
5. Update Applications and Services: Notify applications and services that interact with the `orders` and `order\_items` tables to revert to their previous versions or configurations.
6. Verify System Functionality: Thoroughly test the system to ensure that it is functioning as expected with the rolled-back schema.
7. Post-Rollback Review: Conduct a post-rollback review to identify the root cause of the issue and implement measures to prevent similar problems in the future.

Rollback Preparation

To ensure a smooth rollback, prepare the following:

1. Database Backups: Take a full backup of the database before applying the schema changes.
2. Migration Scripts: Develop and test reverse migration scripts to undo the changes.
3. Data Migration Plans: Document the data migration process and create a plan for reverting data changes.
4. Application and Service Configurations: Save the previous configurations of applications and services that interact with the `orders` and `order\_items` tables.
5. Testing Environments: Set up testing environments to verify the rollback process and ensure that the system functions as expected after the rollback.

By having a well-planned rollback strategy in place, you can quickly respond to issues and minimize the impact on the system and its users.

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.