



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-4-maverick

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

Duration: 53s

Actual run cost: \$0.002320

Projected execution cost: \$1.53/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 3:18 AM EST

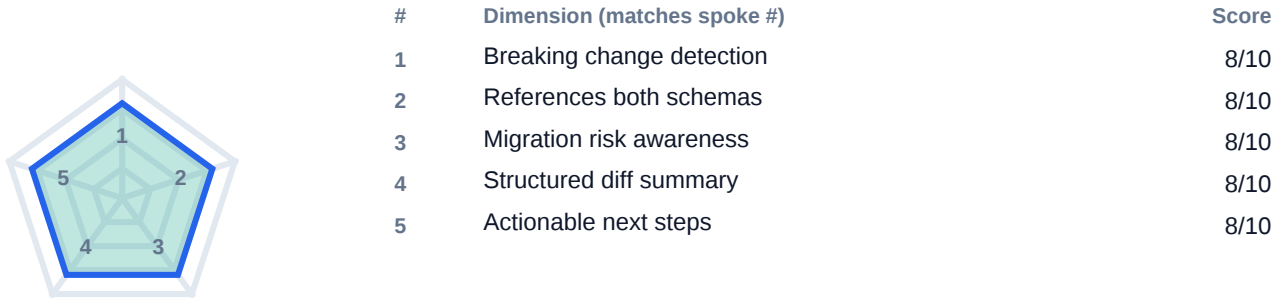
Overall

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: 13486 chars - Latency: 53s - 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: 53s

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

Engineering counterpart

7:17 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

7:18 AM - Responded in 11s

1. Breaking Changes

The following changes between schema v1 (2024-Q1) and schema v2 (2024-Q2) are considered breaking:

1. Type Change:

- In the `orders` table, the data type of `order\_date` has changed from `DATE` to `TIMESTAMP\_NTZ`. This change could affect any code or processes that rely on the specific type or precision of this column.

2. New Required Fields:

- Although the schema does not explicitly denote required fields, the introduction of `order\_item\_id` in `order\_items` and `region\_code` in `orders` could be considered breaking if downstream processes or applications expect the exact schema or require non-null values for these fields. However, based on the provided schema, there's no explicit indication that these fields are required (i.e., not nullable). Still, it's worth noting that any existing INSERT or SELECT statements not accounting for `order\_item\_id` and `region\_code` might need adjustments.

3. Removed Tables/Columns:

- There are no tables or columns removed between v1 and v2. All tables and columns from v1 are present in v2, albeit with some changes and additions.

2. Non-Breaking Additions

The following are non-breaking additions:

1. New Columns:

- `region\_code` has been added to the `orders` table.
- `order\_item\_id` has been added to the `order\_items` table.

2. New Tables:

- The `customers` table has been added.

These additions are considered non-breaking because they do not remove or alter existing schema elements in a way that would directly cause failures in existing applications or processes that are correctly referencing the original schema. However, code adjustments might be necessary to leverage these new schema elements.

3. Migration Risks and Recommended Verification Steps

1. Migration Risks:

- The change in data type for `order\_date` from `DATE` to `TIMESTAMP\_NTZ` could lead to issues if not handled properly, particularly if timestamp values include non-midnight times or if applications are not prepared to handle the potentially broader range of values.
- The introduction of new columns (`region\_code` and `order\_item\_id`) might require adjustments in ETL processes, data loading scripts, or application code that interacts with these tables.
- The addition of the `customers` table, while non-breaking, represents new data that may need to be integrated or considered by downstream processes.

2. Recommended Verification Steps:

- Verify `order\_date` Migration: Check that the migration process correctly converts `DATE` values to `TIMESTAMP\_NTZ`, ensuring no loss of data or unintended changes (e.g., time zone adjustments).
- Validate New Columns: Ensure that `region\_code` in `orders` and `order\_item\_id` in `order\_items` are correctly populated. For `order\_item\_id`, verify that it is unique and properly referenced if it's intended to be a primary key or identifier.
- Review Usage of New Table: Assess whether the new `customers` table is correctly populated and if its data is consistent with expectations. Downstream applications or reports may need to be updated to include this new data.

- Run Comprehensive Test Suites: Execute thorough test suites that cover data ingestion, processing, and reporting to ensure that the schema changes have not introduced any regressions or data inconsistencies.
- Monitor Data Pipelines: Closely monitor data pipelines and dependent applications post-migration for any signs of issues related to the schema changes.

Engineering counterpart
7:18 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 14s

Let's break down the reasoning step by step for the comparison between schema v1 (2024-Q1) and schema v2 (2024-Q2).

Step 1: Identify Changes in Tables

1. Tables in v1: `orders`, `order\_items`
2. Tables in v2: `orders`, `order\_items`, `customers`

- Observation: The `customers` table is new in v2.
- Assumption: The addition of a new table is generally non-breaking if existing applications or processes do not rely on the absence of this table. However, it's worth noting that new dependencies or integrations might be required to leverage this new table.

Step 2: Analyze Changes Within Existing Tables

1. Compare Columns in `orders` Table:

- v1: `order\_id`, `customer\_id`, `order\_date`, `total\_usd`, `status`
- v2: `order\_id`, `customer\_id`, `order\_date`, `total\_usd`, `status`, `region\_code`

- Observation:
 - `order\_date` type changed from `DATE` to `TIMESTAMP\_NTZ`.
 - `region\_code` is a new column.
- Assumption:
 - The type change for `order\_date` could be breaking if applications or processes rely on the specific type or precision.
 - The addition of `region\_code` is non-breaking as it doesn't affect existing columns or their types.

2. Compare Columns in `order\_items` Table:

- v1: `order\_id`, `sku`, `qty`, `unit\_price`
- v2: `order\_item\_id`, `order\_id`, `sku`, `qty`, `unit\_price`

- Observation:
 - `order\_item\_id` is a new column.
- Assumption: The addition of `order\_item\_id` is non-breaking. However, its presence might require adjustments in applications or processes that now need to account for this column.

Step 3: Assess Breaking Changes

1. Type Changes:

- The change from `DATE` to `TIMESTAMP\_NTZ` for `order\_date` in `orders` is considered breaking because it alters the data type, potentially affecting applications or processes that rely on the original type.

2. New Required Fields:

- The schema does not explicitly indicate whether `region\_code` or `order\_item\_id` are required (not nullable). Thus, we assume they are not required based on the information given. However, in practice, whether these are considered breaking changes depends on downstream expectations and constraints.

3. Removed Tables/Columns:

- No tables or columns from v1 are missing in v2, so there are no breaking changes due to removals.

Step 4: Identify Non-Breaking Additions

1. New Columns:

- `region\_code` in `orders`.
- `order\_item\_id` in `order\_items`.

2. New Tables:

- `customers` table.

These are considered non-breaking as they represent additions rather than alterations or removals of existing schema elements.

Step 5: Evaluate Migration Risks and Verification Steps

1. Migration Risks:

- The type change for `order\_date`.
- Potential issues if downstream processes expect or require specific handling of the new columns (`region\_code`, `order\_item\_id`).

2. Verification Steps:

- Validate the `order\_date` type change to ensure data integrity.
- Check the population and consistency of new columns (`region\_code`, `order\_item\_id`).
- Review the usage and integration of the new `customers` table.

Assumptions Made

1. Nullability: Assumed that new columns (`region\_code`, `order\_item\_id`) are nullable since the schema does not specify otherwise.
2. Downstream Dependencies: Assumed

Engineering counterpart
7:18 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

7:18 AM - Responded in 14s

Before shipping the schema changes to production, the following verification steps are recommended:

1. Validate Data Type Changes

- Verify `order\_date` Conversion: Ensure that the conversion from `DATE` to `TIMESTAMP\_NTZ` for `order\_date` in the `orders` table is correctly handled. Check for any potential data loss or corruption, especially around time zone or precision issues.

2. Check New Columns

- `region\_code` in `orders`: Confirm that this column is correctly populated with expected data. Check for any null or unexpected values, especially if there are downstream processes relying on this data.
- `order\_item\_id` in `order\_items`: Verify that this new column is populated correctly, is unique (if intended to be a primary key or identifier), and does not contain null values if it's expected to be not null.

3. Validate New Table

- `customers` Table: Ensure that the new `customers` table is correctly populated with data. Verify data consistency and check for any null or missing values in critical columns like `customer\_id`.

4. Test Data Pipelines and ETL Processes

- Data Ingestion: Test that data ingestion processes correctly handle the new schema, including the new table and columns.
- ETL Processes: Verify that ETL (Extract, Transform, Load) processes are updated and functioning as expected with the new schema changes.
- Data Transformation: Check that any data transformations or calculations involving the changed or new columns are correct.

5. Run Application and Report Tests

- Application Functionality: Execute comprehensive tests of applications that interact with the database to ensure they work correctly with the new schema.
- Reports and Analytics: Validate reports and analytics that depend on the affected tables and columns to ensure they produce the expected results.

6. Monitor for Issues Post-Migration

- Post-Migration Monitoring: Closely monitor the production environment post-migration for any signs of issues related to the schema changes, such as errors, data inconsistencies, or performance impacts.

7. Validate Backup and Recovery Processes

- Backup Processes: Ensure that backup processes are updated to include the new table (`customers`) and are correctly capturing data for the modified tables (`orders`, `order\_items`).
- Recovery Processes: Test recovery processes to ensure that they can correctly restore the database to a consistent state in case of a failure, considering the schema changes.

8. Document Changes and Communicate with Stakeholders

- Documentation: Update documentation to reflect the schema changes, including any changes to data models, ETL processes, or application code.
- Stakeholder Communication: Inform relevant stakeholders about the schema changes, including any potential impacts on

their processes or applications.

By following these verification steps, you can minimize the risk associated with deploying schema changes to production and ensure a smooth transition.

Engineering counterpart
7:18 AM - Responded in 0s

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

Agent response

7:18 AM - Responded in 14s

Top Risks

1. Data Corruption or Loss: The conversion of `order\_date` from `DATE` to `TIMESTAMP\_NTZ` could potentially lead to data corruption or loss if not handled correctly, especially if there are dependencies that are not aware of this change.
2. Application Breakage: Downstream applications or processes that rely on the original schema might break or behave unexpectedly due to the new columns (`region\_code`, `order\_item\_id`) or the changed data type (`order\_date`).
3. Data Inconsistencies: The introduction of new columns and a new table could lead to data inconsistencies if not properly validated and if downstream processes are not updated to handle these changes.
4. Performance Impact: The schema changes could potentially impact database performance, especially if there are significant changes in data volume or if new indexes are required but not implemented.

Rollback Plan

1. Pre-Migration Backup: Ensure a complete backup of the database is taken before making any schema changes. This backup will be crucial for restoring the database in case of a rollback.
2. Rollback Steps:
 - Revert Schema Changes: If possible, revert the schema changes made during the migration. This includes dropping the new `customers` table (if data hasn't been significantly populated or integrated) and removing the new columns (`region\_code`, `order\_item\_id`).
 - Restore `order\_date` Type: If feasible, revert the `order\_date` column type back to `DATE`. This step might be complex and could result in data loss if `TIMESTAMP\_NTZ` values have been modified or if there's data that cannot be converted back to `DATE`.
3. Data Restoration: If significant changes or data modifications have occurred since the migration, restoring from the pre-migration backup might be the most straightforward rollback strategy. This approach ensures data consistency but means losing any changes or updates made post-migration.
4. Application and Process Adjustments: After rolling back the schema changes, adjust applications and processes to once again work with the original schema. This includes reverting any code changes made to accommodate the new schema.
5. Post-Rollback Monitoring: Closely monitor the database and dependent applications post-rollback to ensure that everything is functioning as expected. Be prepared to address any issues that arise.
6. Re-planning and Re-attempt: After a rollback, it's essential to re-evaluate the migration plan, addressing the issues that led to the rollback. This might involve revising the migration strategy, enhancing testing, or implementing additional safeguards before re-attempting the migration.

Mitigation Strategies

To minimize the risk of needing a rollback, consider the following:

- Canary Releases or Gradual Rollouts: Implement schema changes in a controlled manner, such as rolling out changes to a subset of users or data first.
- Extensive Testing: Perform thorough testing of the schema changes in a non-production environment that closely mirrors production.
- Automated Monitoring: Set up automated monitoring to quickly detect any issues post-migration.
- Clear Communication: Communicate the migration plan, potential risks, and rollback strategies to all relevant stakeholders.

By having a well-planned rollback strategy and taking steps to mitigate risks, you can reduce the impact of potential issues arising from the schema migration.

Engineering counterpart

7:18 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.