



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: qwen/qwen3-32b

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

Duration: 1m 53s

Actual run cost: \$0.000872

Projected execution cost: \$0.5755/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 3:23 AM EST

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: 5231 chars - Latency: 113s - 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: 1m 53s

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

Engineering counterpart

7:21 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

7:22 AM - Responded in 39s

Breaking Changes

1. `orders.order\_date` Type Change

- From: `DATE` -> To: `TIMESTAMP\_NTZ`
- Impact: Applications expecting `DATE` values (e.g., `YYYY-MM-DD`) may misinterpret or truncate time components if not updated to handle `TIMESTAMP\_NTZ` (e.g., `YYYY-MM-DD HH:MM:SS`).

2. `order\_items.order\_item\_id` Addition as Required Column

Engineering counterpart

7:22 AM - Responded in 0s

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

Agent response

7:22 AM - Responded in 47s

Step-by-Step Reasoning & Assumptions

1. Breaking Changes

A. `orders.order\_date` Type Change (`DATE` -> `TIMESTAMP\_NTZ`)

- Why it's breaking:

- `DATE` types typically represent calendar dates (`YYYY-MM-DD`).
- `TIMESTAMP\_NTZ` (non-timezone-aware timestamp) includes time components (e.g., `HH:MM:SS`).
- Applications expecting only date values may:
 - Truncate data (drop time components).
 - Fail to parse timestamps if not expecting them.
 - Require changes in downstream systems (e.g., BI tools, reporting).
- Assumption: The schema change impacts interoperability, even if `DATE` was previously stored as ISO 8601 strings. The new schema now enforces stricter timestamp handling.

B. `order\_items.order\_item\_id` Addition as Required Column

- Why it's breaking:

- In v1, the primary key for `order\_items` was `order\_id` (joined to `orders`).
- In v2, a new `order\_item\_id` column is added, which implies a schema shift in primary key logic.
- Applications or ETL pipelines assuming only `order\_id` as a unique identifier for `order\_items` will fail if

Engineering counterpart

7:22 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

7:23 AM - Responded in 15s

Pre-Production Verification Steps and Risks

Here's a detailed checklist based on the schema changes and assumptions:

1. Verify `orders.order\_date` Type Change (`DATE` -> `TIMESTAMP\_NTZ`)

? What to Check

- Data Format Compatibility

- Ensure downstream systems (e.g., reporting tools, APIs, ETL pipelines) can handle `TIMESTAMP\_NTZ` values (e.g., `2024-05-01 14:30:00` vs. `2024-05-01`).
- Confirm that `DATE`-typed data (e.g., `YYYY-MM-DD`) is not inadvertently dropped (e.g., truncation when converting to `TIMESTAMP\_NTZ` or vice versa).

- Historical Data Migration

Engineering counterpart

7:23 AM - Responded in 0s

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

Agent response

7:23 AM - Responded in 13s

Top Risks

1. `orders.order\_date` Type Change

- Risk: Applications or ETLs using `DATE` (e.g., `YYYY-MM-DD`) may fail to process the new `TIMESTAMP\_NTZ` format unless explicitly updated.
- Example Impact: A dashboard expecting `DATE` might error when receiving `2024-04-15 14:30:00` instead of `2024-04-15`.
- Assumption: The data pipeline assumes backward type compatibility (which is not universal).

2. `order\_items.order\_item\_id` Addition as Required Column

- Risk: ETLs or application logic that inserts into `order\_items` without including `order\_item\_id` will fail due to `NOT NULL` constraints.
- Example Impact: Legacy systems writing to `order\_items` via `order\_id` alone will now throw errors.
- Assumption: The column was added without default values or auto-increment logic in the database (i.e., requires explicit population).

3. New Table `customers` Introduced

- Risk: Applications relying on joins to `orders.customer\_id` (e.g., for segmentation) may break if they attempt to query the new `customers` table or its columns (`region`, `segment`) before it's populated.
- Example Impact: A report using `orders.customer\_id` to join to `customers.region` will fail if `customers` is empty.

4. Schema Version Mismatch

- Risk: If `2024-Q2` is deployed while some systems still reference `2024-Q1`, data inconsistencies or failures may occur (e.g., stale data in downstream systems).

Rollback Plan

Option 1: Revert Database Schema to v1

1. `orders` Table

- Recreate the `order\_date` column as `DATE` and drop the `TIMESTAMP\_NTZ` version (if already migrated).
- Risk: If `order\_date` was populated with timestamps in v2, reverting to `DATE` will truncate time components (losing sub-date precision).

2. `order\_items` Table

- Drop the new `order\_item\_id` column only if all existing records in the column are `NULL` (no data has been written yet).
- Risk: If `order\_item\_id` was already populated, dropping it would delete data and break existing dependencies.

3. `customers` Table

- Drop the `customers` table entirely (no data yet).
- Risk: If the table was partially populated, this would delete customer metadata (e.g., `region`, `segment`).

4. Version Reset

- Update the schema version back to `2024-Q1` in metadata (e.g., schema registry, documentation).

Option 2: Shadow the v2 Schema

- Deploy v2 schema as a parallel version (not overwriting v1) and ensure all systems use the correct version until they're ready to migrate.
- When Appropriate: If the change is not yet fully adopted and you're testing in a staging environment.

Option 3: Conditional Rollback with Data Reconciliation

- For `orders.order\_date`:
 - Migrate data from `TIMESTAMP\_NTZ` to `DATE` (e.g., `TRUNC(order\_date, 'DAY')` in SQL).

- Recreate the column as `DATE` and populate the reconciled values.
- For `order\_items.order`

Engineering counterpart

7:23 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.