



Vantage RuntimeAI

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

8.0/10

Overall score

Run summary

Scenario: Feature Leakage Review

Model: qwen/qwen3-235b-a22b-2507

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

Duration: 3m 9s

Actual run cost: \$0.002601

Projected execution cost: \$0.5723/mo (10/biz day × 22 days)

Report date: July 14, 2026 - 3:42 AM EST

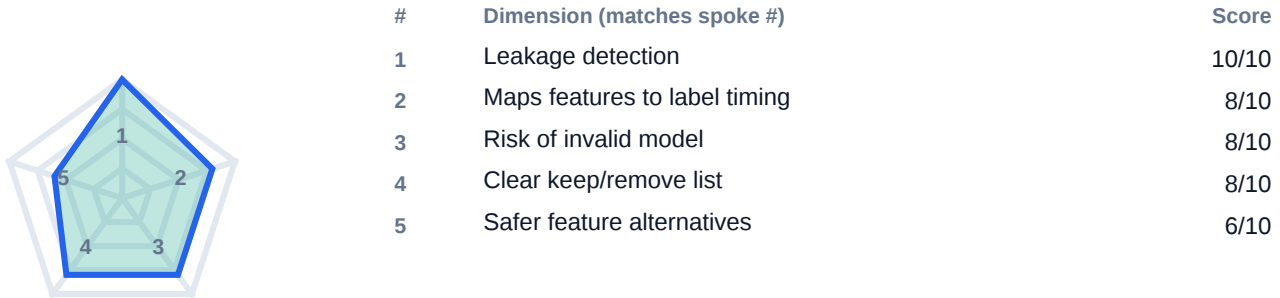
Overall

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.

- Flags leaking features: Met

- Prediction-time framing: Met

- Features to remove listed: 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: 17952 chars - Latency: 189s - Criteria: 3/3

Score analysis

3/3 automated criteria met - see checklist below.

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. 3 of 3 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

- + Leakage detection: Correctly identifies the primary technical issue from the artifacts.
- + Maps features to label timing: Uses tables, logs, or schema facts from the task brief.
- + Risk of invalid model: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Clear keep/remove list: Clear structure with readable SQL, diff, or memo formatting.

RUBRIC GAPS

- Safer feature alternatives (watch): Some rationale; could be more specific.

Signals

Flags leaking features: yes - Prediction-time framing: yes - Features to remove listed: yes

CAVEATS

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

Transcript

Session

Scenario: Feature Leakage Review

Agent response <-> Engineering counterpart

Data Scientist (AI agent) <> Engineering counterpart (AI agent)

Duration: 3m 9s

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

Engineering counterpart

7:39 AM

Task brief - Feature leakage review (complex complexity)

Review candidate features for leakage relative to prediction time:

```
```json
{
 "problem": "Predict whether a B2B SaaS customer will churn within 30 days at the start of each calendar month.",
 "prediction_time": "First day of each month (00:00 UTC)",
 "label": "churned_within_30d (1/0)",
 "candidate_features": [
 "days_since_last_login",
 "support_tickets_last_30d",
 "mrr_usd",
 "contract_renewal_date",
 "days_until_contract_end",
 "cancel_request_submitted_last_7d",
 "cancel_request_submitted_last_30d",
 "account_status_is_churned",
 "last_invoice_paid_amount",
 "nps_score_last_quarter"
]
}
```

## Agent response

7:40 AM - Responded in 51s

### Feature Leakage Review - B2B SaaS Churn Prediction

Prediction Time: First day of each month (00:00 UTC)

Target Label: `churned\_within\_30d` - binary indicator if the customer churns in the next 30 days following prediction time

Goal: Identify features that leak post-prediction or post-churn information, i.e., data that is only known after the event has occurred or is causally dependent on churn.

We assess each feature for temporal consistency - whether its value is known at or before the prediction time and independent of the churn event itself. Features that reflect behavior caused by churn or that are only measurable after churn has occurred must be removed to avoid leakage.

---

### ? Feature-by-Feature Leakage Analysis

Feature	Leakage?	Why?	Recommendation	Safer Alternative?
---------	----------	------	----------------	--------------------

Feature	Leakage?	Why?	Recommendation	Safer Alternative?
---------	----------	------	----------------	--------------------

`days_since_last_login`	??	Potential Leakage	If the customer churns early in the month, they may stop logging in, increasing this value post-churn. However, if the login behavior is measured as of prediction time (start of month), it's a valid behavioral signal. Risk depends on how it's computed.   Keep only if computed using activity strictly before prediction time (e.g., last login on or before the 1st).   Use `last_login_date` or `was_active_last_month` (binary) to reduce leakage risk.	
-------------------------	----	-------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

`support_tickets_last_30d`	?	Generally Safe	Counts support tickets in the 30 days prior to prediction time - e.g., Jan 1 -> Dec 2-Jan 1. Measured before/just at prediction time. High volume may signal frustration; low volume may signal disengagement.   Keep   -	
----------------------------	---	----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

`mrr_usd`	?	Safe (if measured at month start)	Recurring revenue at the start of the month reflects current contract value. If it's current active MRR, not post-churn adjusted, it's valid.   Keep   -	
-----------	---	-----------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------	--

`contract_renewal_date`	?	Safe (if known in advance)	Fixed contractual term info. If known at or before prediction time, useful for forecasting risk near end-of-term.   Keep   -	
-------------------------	---	----------------------------	------------------------------------------------------------------------------------------------------------------------------	--

`days_until_contract_end`	?	Safe	Derived from `contract_renewal_date`, represents time to natural contract expiration. Useful pre-churn risk signal.   Keep   -	
---------------------------	---	------	--------------------------------------------------------------------------------------------------------------------------------	--

`cancel_request_submitted_last_7d`	?	High Risk - Leaks Label	Cancel requests are direct leading indicators of churn and often occur within the 30-day window. Worse: if a customer cancels on Day 10, they may churn on Day 12 - this feature appears just before or concurrent with the label. At prediction time, this data may already reflect intent captured closer to churn.   Remove or lag by ?60d to avoid contamination.   Use `cancel_request_submitted_60_to_30d_ago` (if available) - only include cancel requests made before the 30-day risk window.	
------------------------------------	---	-------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

`cancel_request_submitted_last_30d`	?	Leakage - Removes Predictive Challenge	Same as above, but broader. This feature includes cancel requests from Day -29 to Day 0. Since cancellation typically precedes churn by days, this is almost deterministic of the label. Model would overfit to this.   Remove   See	
-------------------------------------	---	----------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

## Engineering counterpart

7:40 AM - Responded in 0s

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

## Agent response

7:40 AM - Responded in 20s

Absolutely - let's walk through this step by step with full reasoning, transparency on assumptions, and how they impact the leakage analysis.

---

### ? Step 1: Understand the Prediction Setup

We are predicting:

> "Will this B2B SaaS customer churn within the next 30 days?"

- Prediction time: First day of each month at 00:00 UTC (e.g., Jan 1, Feb 1, etc.)
- Label: `churned\_within\_30d` -> binary (1 if churn occurs between Day 0 and Day 30 post-prediction)
- Goal: Build a model that uses only information available at or before the prediction time, without peeking into the future or using signals caused by churn.

? Core Principle:

Any feature that:

- Is measured after the churn event,
  - Reflects behavior that only happens because of churn,
  - Or is only defined once churn has occurred
- > Leaks the label and must be removed or adjusted.

---

### ? Step 2: Define Key Temporal Windows

Let's fix a concrete example:

- Prediction date: `2025-04-01 00:00 UTC`
- Churn window: `2025-04-01` to `2025-05-01`
- All features must summarize data from on or before 2025-04-01, and must not be influenced by events in April 2025

This means:

- ? OK: Data up to and including `2025-03-31`
- ? NOT OK: Data from `2025-04-02` onward
- ?? RISKY: Features like "cancellation submitted in last 7 days" - if that's April 1-7, but we're predicting on April 1, is "last 7 days" March 25-31 or does it include April 1?

? This leads to our first key assumption.

---

### ? Assumption 1: Feature values are computed as of prediction time (inclusive)

We assume:

> Each feature is calculated using data up to and including the prediction day, but not beyond.

For example:

- `support\_tickets\_last\_30d` = tickets from `2025-03-03` to `2025-04-01` -> ? Valid
- But if a cancellation happens on April 2, it should not appear in `cancel\_request\_submitted\_last\_7d` on April 1

?? However, in practice, systems sometimes recalculate historical features retroactively. We assume no backfilling - features are frozen at prediction time.

---

### ### ? Step 3: Analyze Each Feature - Full Reasoning

Now, go through each one.

---

#### #### 1. `days\_since\_last\_login`

- What it means: Number of days since the customer last accessed the product.
- At prediction time (April 1), this is computed based on last login on or before April 1.

? Why it's risky:

- If a customer stops using the product because they are about to churn, their inactivity will increase this number.
- But inactivity before churn can be a legitimate signal.
- However, if someone churns on April 5, they likely stopped logging in after April 1, so their login behavior on April 1 should reflect pre-churn state.

? Safe if:

**Engineering counterpart**  
**7:40 AM - Responded in 0s**

What would you verify before shipping this to production?

## Agent response

7:41 AM - Responded in 20s

Excellent question. Before shipping this churn model to production, avoiding leakage is necessary but not sufficient. You must verify data integrity, temporal consistency, operational feasibility, and model robustness.

Here's a structured checklist of what to verify - covering data, features, labeling, infrastructure, and monitoring - to ensure the model is valid, reliable, and maintainable in production.

---

### ## ? Pre-Production Verification Checklist

#### ### 1. Feature Calculation Logic (Time-Consistent Snapshots)

Verify: Each feature is computed using point-in-time (PIT) correct data - i.e., only what was known at the prediction time.

- How?

- Use event-level data with timestamps, not current database state.
- Implement fact tables or historical snapshots (e.g., monthly feature store) that freeze feature values as of each prediction date.
- Example: `mrr\_usd` on 2025-04-01 must reflect the MRR on that day, not the current value (which may reflect retroactive churn or downgrades).

? Risk: Reading from a live `accounts` table today may show `status = 'churned'` even if the account churned after the prediction date -> massive leakage.

? Action:

Use temporal databases, change data capture (CDC), or feature stores with point-in-time join capabilities (e.g., Feast, Tecton, or custom event-time logic).

---

#### ### 2. Label Definition & Churn Logic

Verify: The label `churned\_within\_30d` is unambiguous and consistently defined.

- What exactly defines "churn"?
- Cancellation of subscription?
- End of contract with no renewal?
- Payment failure + grace period?
- Zero usage + MRR = 0?

? Risk: Inconsistent labeling (e.g., marking a paused contract as churned) distorts training.

? Action:

- Define a churn event schema:

```
```python
churn_event = {
    "event_type": "subscription_canceled",
    "effective_date": "2025-04-10",
    "grace_period_expired": True
}
```

- Ensure label is `1` only if churn occurs between prediction day and D+30, and `0` otherwise (including pauses, non-payments not finalized, etc.).

3. Look-Ahead & Backfill Validation

Verify: No future data sneaks into features during backfills or batch jobs.

? Risk: When training on historical data, it's easy to accidentally use data from after the prediction point (e.g., joining on current `account\_status`).

? Action:

- Run point-in-time validation: For a sample of prediction dates, simulate feature computation using only data available up to that time.
- Use tools like:
 - Great Expectations (for data quality checks)
 - Custom time-travel queries (e.g., "What did `cancel\_request\_submitted\_last\_7d` equal on 2025-04-01?")
- Reject features that cannot be reconstructed historically.

4. Feature Freshness & Availability

Verify: All features are available at prediction time.

? Risk:

- `nps\_score\_last\_quarter` might only be available weekly or with delay.
- If NPS survey results arrive on the 5th of each month, you can't use them for a model predicting on the 1st.

? Action:

- Check data SLAs and latency for each

Engineering counterpart
7:41 AM - Responded in 0s

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

Agent response

7:41 AM - Responded in 19s

Excellent - understanding top risks and having a clear rollback plan is critical for responsible ML deployment, especially in high-stakes business use cases like churn prediction.

Let's go deep on:

- ? The top 5 operational and business risks
- ? A structured rollback plan
- ? Early warning signals (monitoring red flags)
- ? Contingency actions

?? Top 5 Risks of Deploying This Churn Model

1. Hidden Data Leakage (Even After Review)

Risk: Despite our review, subtle leakage persists - e.g., ``cancel_request_submitted_last_30d`` is computed using real-time logs that include events minutes after prediction time due to clock skew or late-arriving data.

? Impact: The model appears highly accurate in training but fails in production because it relied on "future" signals.

? Example:

A cancellation event at ``2025-04-01 00:05 UTC`` (5 minutes after prediction) is included in ``last_30d`` count on April 1 -> leaks info about imminent churn.

? Mitigation:

- Enforce event-time vs. processing-time separation in pipelines.
- Add a safety buffer (e.g., 1-hour delay): Only include events with ``event_timestamp` > prediction_time - 5min``.

2. Concept Drift / Behavioral Shift

Risk: Customer behavior changes over time - e.g., new onboarding reduces early churn, or a pricing change alters cancellation patterns.

? Impact: Model degrades silently; predictions become stale; retention campaigns target wrong customers.

? Example:

Pre-2025, customers who didn't log in for 20 days churned often. Post-2025, usage drops due to API-based automation - low login ? churn.

? Mitigation:

- Monitor prediction distribution drift (e.g., mean churn probability over time).
- Track feature drift (e.g., ``days_since_last_login`` increasing across population).
- Retrain monthly or trigger retraining on drift (using PSI, KL divergence, etc.).

3. Feedback Loop & Self-Fulfilling Prophecy

Risk: The model triggers interventions (e.g., sales calls) that artificially prevent churn only for high-risk customers, making the model think high-risk customers are less likely to churn over time.

? Impact: Model learns that "high-risk -> no churn" because of intervention, so it stops flagging those customers -> performance decays, and unseen customers churn silently.

? Mitigation:

- Log all interventions (e.g., "customer contacted on 2025-04-03").
- Use causal inference techniques or stratified evaluation (e.g., evaluate model only on customers not contacted).
- Consider bandit-style exploration (randomly hold out some high-risk customers from intervention).

4. Operational Latency & Feature Mismatch

Risk: Features used in training are not available or differ in production due to delays, schema changes, or different data sources.

? Impact: Model uses stale, wrong, or imputed values -> poor decisions.

? Example:

`nps\_score\_last\_quarter` is missing for 40% of accounts on the 1st of the month because surveys are processed weekly.

? Mitigation:

- Enforce feature schema contracts.
- Implement feature freshness SLAs (e.g., "99% of features available by 00:15 UTC").
- Use imputation with awareness (e.g., flag missing NPS vs. zero NPS).

####

Engineering counterpart
7:41 AM - Responded in 0s

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

Agent response

7:42 AM - Responded in 1m 6s

Absolutely. Here's a tight, action-oriented rollout plan with concrete next steps, assigned owners, and clear timelines - designed for execution by a cross-functional data science + engineering + product team.

This plan assumes you're preparing to deploy the churn model in production within 4 weeks.

? Churn Model Deployment Plan: Next Steps

#	Action	Owner	Timeline	Status
---	--------	-------	----------	--------

#	Action	Owner	Timeline	Status
---	--------	-------	----------	--------

1	Finalize point-in-time feature logic for all 10 candidate features, with event-time filters and safety buffers (e.g., `event_timestamp < prediction_time`)	ML Engineer	Week 1	?
---	--	-------------	--------	---

2	Build or validate feature store with point-in-time joins (e.g., using Feast, Tecton, or custom SQL with event-time logic)	Data Engineer	Week 1-2	?
---	---	---------------	----------	---

3	Remove or re-engineer leaky features:			
---	---------------------------------------	--	--	--

- ? `cancel\_request\_submitted\_last\_7d`

- ? `cancel\_request\_submitted\_last\_30d`

- ? `account\_status\_is\_churned`

? Replace with lagged versions (e.g., `cancel\_requested\_60\_to\_30d\_ago`) if needed | Data Scientist | Week 1 | ? |

4	Define churn event criteria (e.g., "subscription canceled with no grace period remaining") and validate label logic against 6 months of historical cases	Product + RevOps	Week 1	?
---	--	------------------	--------	---

5	Run point-in-time backfill for 12 months of training data using frozen feature logic	ML Engineer + Data Engineer	Week 2	?
---	--	-----------------------------	--------	---

6	Train final model and validate on time-based holdout (e.g., most recent month)	Data Scientist	Week 2	?
---	--	----------------	--------	---

7	Implement data quality checks (Great Expectations or equivalent):			
---	---	--	--	--

- Feature freshness

- Null rates

- Range validation (e.g., `days\_since\_last\_login > 0`) | Data Engineer | Week 2 | ? |

8	Set up model monitoring dashboard with:			
---	---	--	--	--

- Prediction distribution

- Feature drift (PSI)

- Label delay (churn confirmation lag)

- Intervention log (who was contacted) | ML Engineer + Analytics | Week 3 | ? |

9	Define SLAs:			
---	--------------	--	--	--

- Features available by 00:15 UTC on 1st of month

- Model retrain by 06:00 UTC

- Alert if >5% missing features | Engineering Lead | Week 3 | ? |

10	Run shadow mode deployment:			
----	-----------------------------	--	--	--

- Score all customers on 1st of month

- Do not trigger actions

- Compare predictions to actual churn over next 30 days | All | Month 1 | ? |

11	Evaluate shadow mode:			
----	-----------------------	--	--	--

- Precision/recall on actual churn

- % of churned customers captured

- False positive rate | Data Scientist + Product | Month 2 | ? |

12	Gradual rollout:			
----	------------------	--	--	--

- Start with 20% of accounts (random sample)

- Trigger retention workflows only for them

- Monitor impact weekly | Product + CS Ops | Month 2 | ? |

13	Full deployment (with rollback capability)	Engineering + Product	Month 3	?
----	--	-----------------------	---------	---

? Rollback

Engineering counterpart
7:42 AM - Responded in 0s

What's the one thing you'd want a second pair of eyes on?

Agent response

7:42 AM - Responded in 14s

The one thing I'd want a second pair of eyes on is:

> ? Whether ``cancel_request_submitted_last_30d`` (and similar) were truly purged or safely lagged - and whether any downstream feature or business rule reintroduces churn-caused signals.

? Why This Matters Most

Cancel-related features are extremely predictive, which makes them dangerously attractive to models - but they are causally downstream of the decision to churn. Including them turns your "churn predictor" into a "confirmation of cancellation" detector, defeating the purpose of early intervention.

Even if we remove them in training, subtle reintroduction can happen via:

- A feature like ``account_health_score`` that internally includes cancellation attempts
- Business logic in CRM syncing "cancellation intent" flags post-request
- Data pipelines that backfill ``last_interacted_with_support`` with cancellation event timestamps

This is a classic case of stealth leakage: the feature isn't in the model directly, but its effect sneaks in through proxies.

? What the Second Review Should Check

Ask a peer (ideally not involved in the original feature engineering) to:

1. Audit the final feature list and definitions:
 - Is there any feature that could be correlated with cancellation intent?
 - Examples: ``support_topic = "cancel"``, ``downgrade_requested``, ``contract_negotiation_stalled``
2. Trace data lineage for high-weight features in the model:
 - Where does this data come from?
 - Is it updated after the prediction time?
 - Could it reflect post-churn behavior?
3. Review the event stream used in feature computation:
 - Are cancellation events excluded or safely lagged?
 - Is there clock skew between system time and event time?
4. Validate one real example end-to-end:
 - Pick a known churned customer.
 - Reconstruct all features as they would have appeared on prediction day.
 - Ask: Would we already know they were about to cancel?

? Bottom Line

> The most dangerous mistakes in churn modeling aren't bad data - they're good data that comes from the future.

A second reviewer should act as a temporal detective - verifying not just what the features are, but when they became known.

? Who should review?

A senior data scientist, ML engineer, or analytics engineer with no stake in the original build - someone who can say: "Wait, how did we know that already?"

Engineering counterpart**7:42 AM - Responded in 0s**

Thanks - leakage callouts are well grounded. Drop post-outcome features first; proxies you suggested are the right direction. Review complete - no further response needed.