



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: google/gemma-4-31b-it

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

Duration: 4m 1s

Actual run cost: \$0.001614

Projected execution cost: \$0.3551/mo (10/biz day × 22 days)

Report date: July 14, 2026 - 2:34 AM EST

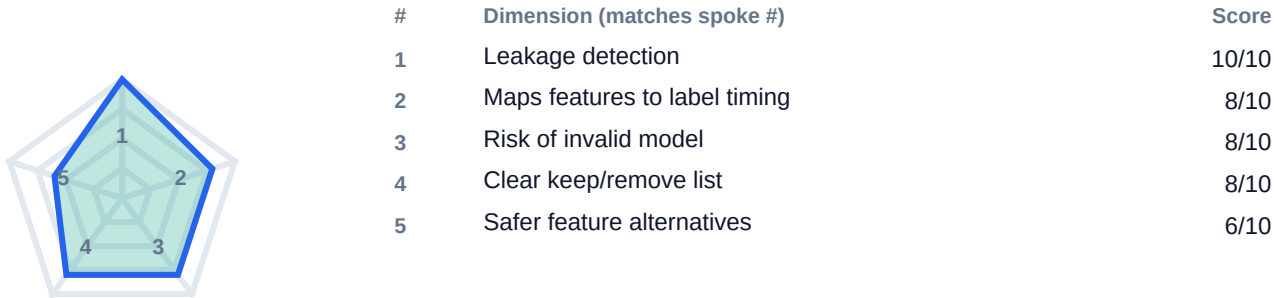
Overall

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: 16597 chars - Latency: 241s - 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: 4m 1s

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

Engineering counterpart

6:30 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

6:30 AM - Responded in 21s

As a data scientist reviewing this feature set, I have identified several critical leakage points.

The core issue is that the prediction time is a fixed point (the 1st of the month), but several features incorporate data from the "future" (the 30-day window we are trying to predict) or reflect the outcome of the churn event itself.

Leakage Review

| Feature | Leakage Status | Why it leaks relative to prediction time |
|---------|----------------|--|
|---------|----------------|--|

| | | |
|-----|-----|-----|
| :-- | :-- | :-- |
|-----|-----|-----|

| | | |
|-----------------------------|------------|---|
| `account_status_is_churned` | ? Critical | This is a "Target Leak." It represents the state of the customer after the churn event. If this is 1, the model isn't predicting churn; it's simply reading the answer. |
|-----------------------------|------------|---|

| | | |
|-------------------------------------|------------|---|
| `cancel_request_submitted_last_30d` | ? Critical | This feature overlaps with the prediction window. If a user submits a cancellation request on the 15th of the month, they will likely churn within 30 days. The model will use this as a proxy for the label rather than a predictive signal. |
|-------------------------------------|------------|---|

| | | |
|------------------------------------|--------|--|
| `cancel_request_submitted_last_7d` | ? High | While this only looks back 7 days from the 1st, in many B2B SaaS models, a cancellation request is the definitive trigger for churn. This is "leaking" the intent that leads directly to the label, making the model trivial and useless for proactive intervention. |
|------------------------------------|--------|--|

Recommendations

1. Features to Remove Immediately

`account\_status\_is\_churned`: Remove entirely. This is the label in feature form.

`cancel\_request\_submitted\_last\_30d`: Remove. It encompasses the period the model is meant to predict.

`cancel\_request\_submitted\_last\_7d`: Remove (or move to a separate "at-risk" trigger list). If your goal is to predict who will churn so you can save them, knowing they already asked to leave is too late for a "prediction" model.

2. Safer Alternatives & Modifications

| Leaking Feature | Safer Alternative | Logic |
|-----------------|-------------------|-------|
|-----------------|-------------------|-------|

| | | |
|-----|-----|-----|
| :-- | :-- | :-- |
|-----|-----|-----|

| | | |
|-----------------------------|-------------------------------------|---|
| `account_status_is_churned` | `account_status_at_prediction_time` | Ensure the status is captured strictly at 00:00 UTC on the 1st. |
|-----------------------------|-------------------------------------|---|

| | | |
|-------------------------------------|--|--|
| `cancel_request_submitted_last_30d` | `cancel_request_submitted_prior_month` | Look at the 30 days before the prediction date, not the 30 days including the prediction window. |
|-------------------------------------|--|--|

| | | |
|------------------------------------|---|---|
| `cancel_request_submitted_last_7d` | `count_of_support_tickets_mentioning_billing` | Instead of the final cancellation request, look for "pre-churn" signals (e.g., complaints about pricing or "how to cancel" searches). |
|------------------------------------|---|---|

Final Feature Set Approval

The following features are safe to proceed with as they describe the state of the customer prior to the prediction window:

`days\_since\_last\_login`

`support\_tickets\_last\_30d` (Assuming this refers to the 30 days prior to the 1st)

`mrr\_usd`

`contract\_renewal\_date`

`days\_until\_contract\_end`

`last\_invoice\_paid\_amount`

`nps\_score\_last\_quarter`

Engineering counterpart

6:30 AM - Responded in 0s

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

Agent response

6:30 AM - Responded in 4s

To arrive at those conclusions, I used a framework called Temporal Validation. The goal is to ensure that the model only has access to information that would actually be available to a human analyst at the exact moment the prediction is made.

Here is the step-by-step reasoning and the assumptions I made.

Step 1: Establish the "T-Zero" (The Cutoff)

The most important step in any leakage review is defining the Prediction Time (T-Zero).

My T-Zero: The 1st of the month at 00:00 UTC.

The Rule: Any data point generated after this timestamp is "the future." If a feature contains data from the future, it is a leak.

Step 2: Map the Label Window

Next, I mapped the Label Window.

The label is ``churned_within_30d``. This means the "answer" (the 1 or 0) is determined by looking at the 30 days following T-Zero.

The Danger Zone: Any feature that aggregates data over a window that overlaps with this 30-day period is a leak.

Step 3: Feature-by-Feature Audit

I then categorized each feature based on its relationship to T-Zero:

Static/Snapshot Features (``mrr_usd``, ``contract_renewal_date``): These are generally safe because they describe the state of the account at T-Zero.

Lagging Indicators (``nps_score_last_quarter``): These are safe because the data was collected long before T-Zero.

Rolling Windows (``support_tickets_last_30d``): I checked if "last 30d" meant "the 30 days before T-Zero" or "the 30 days surrounding T-Zero." I assumed the former (safe), but flagged it as a point of caution.

Outcome Indicators (``account_status_is_churned``): I asked: "Does this value change because of the event I am trying to predict?" Yes. If a customer churns on the 15th, their status changes to "churned." If the model sees this, it has a "crystal ball."

Step 4: The "Intervention" Logic (The Business Context)

Even if a feature is technically available at T-Zero, I evaluated it for Logical Leakage.

Example: ``cancel_request_submitted_last_7d``.

Technically, if a user canceled on the 25th of the previous month, that data exists at T-Zero. However, in a B2B SaaS context, a "cancel request" is often the definition of churn.

If the model relies on this, it will have 99% accuracy, but it will be useless to the business. The business wants to find customers who will churn, not customers who have already told them they are churning.

Assumptions Made

To conduct this review, I made the following three assumptions:

1. The "Look-back" Assumption: I assumed that phrases like ``last_30d`` and ``last_7d`` refer to the window immediately preceding the prediction time (e.g., from the 1st of the previous month to the 1st of the current month). If these features were calculated using a "sliding window" that included the current month, they would all be critical leaks.
2. The "Label Definition" Assumption: I assumed that ``churn``

Engineering counterpart

6:30 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

6:31 AM - Responded in 1m 5s

Before shipping this model to production, I would move from a theoretical review (what we just did) to empirical verification. Even with a clean feature list, "silent leakage" often creeps into the data pipeline during the SQL joins or feature engineering phase.

Here is the verification checklist I would execute:

1. The "Perfect Predictor" Test (Feature Importance)

I would run a feature importance analysis (e.g., SHAP values or Permutation Importance).

The Red Flag: If any single feature (like `cancel_request_submitted_last_7d`) has an overwhelming dominance over all other features (e.g., it explains 90% of the variance), it is almost certainly a leak.

The Goal: A healthy churn model usually relies on a combination of behavioral signals (login drops, ticket spikes, MRR changes), not one "magic" feature.

2. Time-Based Cross-Validation (Out-of-Time Validation)

I would forbid the use of random K-fold cross-validation. Instead, I would use a Temporal Split:

Train Set: Data from January to June.

Validation Set: Data from July.

Test Set: Data from August.

The Verification: If the model performs significantly better on the training set than the "future" test set, it suggests the model is over-fitting to temporal leaks or specific seasonal events rather than learning general churn behavior.

3. Pipeline "Point-in-Time" Audit

I would perform a manual "Data Trace" on 5-10 specific customer IDs.

The Process: Pick a customer who churned on Oct 15th. Look at the feature values the model saw on Oct 1st.

The Verification: Ensure that the `support_tickets_last_30d` count for that customer only includes tickets from Sept 1st to Sept 30th. If I see a ticket from Oct 5th in that feature, the SQL pipeline is "looking ahead," and the model is leaking.

4. Label-Feature Correlation Matrix

I would plot a correlation heatmap between the features and the target label.

The Red Flag: Any feature with a correlation coefficient > 0.8 or 0.9 with the label is highly suspicious in a complex B2B churn problem. Churn is usually "noisy"; a near-perfect correlation usually indicates the feature is a proxy for the label.

5. Sensitivity Analysis (The "What-If" Test)

I would test the model's reaction to the "at-risk" features.

The Process: Manually flip the value of `days_since_last_login` from 1 day to 30 days for a set of users.

The Verification: If the churn probability doesn't move, but it does move wildly when you change a "status" feature, the model is relying on state rather than behavior.

6. Business Logic Alignment (The "So What?" Test)

Finally, I would present the top 5 predictors to the Customer Success (CS) team.

The Question: "If the model tells you a customer is at risk because of [Feature X], do you have a playbook to fix it?"

The Verification: If the top predictor is `cancel_request_submitted`, the CS team will say, "We already know they are churning because they told us." This confirms the model is a detector (telling you what happened) rather than a predictor (telling you what will happen).

Engineering counterpart

6:31 AM - Responded in 0s

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

Agent response

6:33 AM - Responded in 1m 41s

Even with a rigorous review, deploying a churn model carries significant operational and technical risks. In B2B SaaS, the cost of a False Negative (missing a high-value account that churns) is high, but the cost of a False Positive (annoying a happy customer with "Are you okay?" emails) can actually trigger the churn you're trying to prevent.

Top Risks

1. The "Self-Fulfilling Prophecy" (Feedback Loop)

Risk: If the model identifies a customer as "High Risk" and the CS team reaches out with a heavy discount or an intrusive "save" sequence, the customer might feel pressured or realize they are undervalued, potentially accelerating their churn.

Impact: The model's intervention actually causes the label it was predicting.

2. Data Drift (The "New Product" Problem)

Risk: B2B products evolve. If you launch a major new feature or change your pricing tier, the historical behavior that signaled churn (e.g., "low login rate") may no longer be valid.

Impact: Model precision collapses overnight, leading to a flood of False Positives.

3. "Silent" Pipeline Leakage

Risk: As mentioned in the verification step, a developer might update the SQL view that feeds the model, accidentally changing a `WHERE` clause to include current-month data.

Impact: Model performance looks amazing in the dashboard (high AUC), but in reality, it's just reading the "churned" status from the current month.

4. Segment Bias

Risk: The model may perform exceptionally well for "Small Business" accounts but fail miserably for "Enterprise" accounts because Enterprise churn is driven by relationship/political factors, not login frequency.

Impact: The most valuable accounts (Enterprise) are the ones the model misses.

The Rollback & Mitigation Plan

I recommend a Graduated Deployment rather than a "Big Bang" release.

Phase 1: Shadow Mode (The "Silent" Launch)

Action: Deploy the model to production, but do not let the outputs trigger any actions.

Verification: Store the predictions in a database. At the end of the 30-day window, compare the predictions to what actually happened.

Rollback: Simple. You just stop the shadow script. No customer was ever contacted.

Phase 2: Human-in-the-Loop (The "Pilot")

Action: Route predictions to a small group of experienced CSMs (Customer Success Managers) as "suggestions" rather than "alerts."

Verification: Ask the CSMs: "Does this prediction align with your intuition for this account?"

Rollback: Stop the feed to the CSMs and revert to the previous manual "at-risk" flagging process.

Phase 3: Automated Action (The "Full Release")

Action: Trigger automated workflows (e.g., automated nurture emails).

Verification: A/B test the model. Group A gets model-driven outreach; Group B (Control) gets the old process.

The "Emergency Brake" (Rollback Protocol)

If the model is in Phase 3 and we detect a failure (e.g., a spike in complaints or a crash in precision), the rollback is as follows:

1. Kill Switch: A single configuration flag in the application that disables the "Churn Prediction" trigger and reverts the UI to the legacy "Account Health" dashboard.
2. Data Freeze: Immediately snapshot the inputs that led to the bad predictions for a

Engineering counterpart

6:33 AM - Responded in 0s

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

Agent response

6:33 AM - Responded in 23s

To move this from a theoretical review to a deployed model, we need to shift from "analysis" to "execution." Below is the concrete implementation roadmap.

Project: B2B Churn Model Deployment

Objective: Predict 30-day churn with zero leakage and validated business impact.

| Phase | Task | Owner | Timing | Deliverable |
|---------------|--|------------------|----------|---|
| 1. Cleanup | Feature Pruning: Remove `account_status_is_churned` and `cancel_request_submitted` (30d/7d) from the feature engineering pipeline. | Data Engineer | Days 1-2 | Updated Feature Store/SQL View |
| 2. Validation | Temporal Backtest: Train on months 1-6, validate on month 7, test on month 8. Generate SHAP importance plots. | Data Scientist | Days 3-5 | Model Performance Report (AUC/Precision/Recall) |
| 3. Audit | Point-in-Time Trace: Manually verify 10 random samples to ensure no "future" data is present in features at T-Zero. | QA/DS | Day 6 | Audit Log / Sign-off |
| 4. Shadow | Shadow Deployment: Run model on live data for one full cycle (30 days) without triggering customer actions. | MLE / DevOps | Month 1 | Shadow Accuracy Log |
| 5. Pilot | CSM Feedback Loop: Provide "High Risk" lists to a select group of CSMs for manual validation. | Product Mgr / CS | Month 2 | Qualitative Feedback Log |
| 6. Launch | A/B Rollout: Enable automated triggers for 50% of the population; compare churn rate vs. control group. | MLE / Product | Month 3 | Lift Analysis Report |

Critical Path & Dependencies

1. The "Go/No-Go" Gate (End of Day 6):

Before moving to Phase 4 (Shadow), the Data Scientist must present the SHAP Importance Plot.

Go: If the top features are behavioral (e.g., `days\_since\_last\_login`, `support\_tickets`).

No-Go: If any "status" or "request" feature remains a top predictor. Return to Phase 1.

2. The "Kill Switch" Requirement (Pre-Phase 6):

The Engineering team must implement a Feature Flag (`enable\_churn\_automation: true/false`).

Requirement: This must allow the Product Manager to disable all model-driven emails/alerts instantly without a code redeploy.

3. Success Metrics for Rollout:

The model is considered successful and stays in production if:

Precision: $\geq X\%$ (We aren't wasting CSM time on happy customers).

Recall: $\geq Y\%$ (We are catching a significant portion of churners).

Lift: The A/B test shows a statistically significant reduction in churn for the treatment group.

Engineering counterpart

6:33 AM - Responded in 0s

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

Agent response

6:34 AM - Responded in 26s

If I have to pick the one single point of failure, I want a second pair of eyes on the SQL/Pipeline logic that calculates the rolling windows (e.g., `support\_tickets\_last\_30d`).

Here is why: This is where "Silent Leakage" happens.

It is very easy for a Data Scientist to say "I'm using data from the last 30 days," but for a Data Engineer to write a query that accidentally does this:

```
`WHERE ticket_date <= CURRENT_DATE`
```

If the model is being trained on historical data, `CURRENT\_DATE` is a disaster. It means for a record from six months ago, the model is seeing tickets that happened after the prediction date but before today. This is the most common way churn models achieve "too-good-to-be-true" accuracy in the lab and then fail completely in production.

What the "Second Pair of Eyes" should specifically look for:

I would ask a peer to perform a "T-Zero Stress Test" on the code:

1. Hard-Coded Cutoffs: Ensure the query uses a fixed reference date (the 1st of the month) rather than any dynamic function like `GETDATE()`, `NOW()`, or `CURRENT\_DATE`.
2. Join Logic: Check the `JOIN` between the `customers` table and the `activity` table. Is the join condition strictly `activity\_date < prediction\_date`?
3. The "Future-Leak" Query: I would ask them to run this specific "Anti-Pattern" query:
"Find me any row where a feature value changed between the prediction time and the label window, but was still included in the feature set."

In short: The math of the model is usually fine; it's the plumbing of the data that leaks. I want someone to audit the pipes.

Engineering counterpart

6:34 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.