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-3-12b-it
Roles: Data Scientist (AI agent) <> Engineering counterpart (AI agent)
Duration: 1m 27s
Actual run cost: $0.000765
Projected execution cost: $0.1683/mo (10/biz day × 22 days)
Report date: July 13, 2026 - 11:32 PM EST

Session: sample-lb-rankings-seed-4d9a9cb3-89d9-4e4f-ae14-e632d49c6d06

ds_feature_leakage_v1 - agent - openrouter - google/gemma-3-12b-it - ended

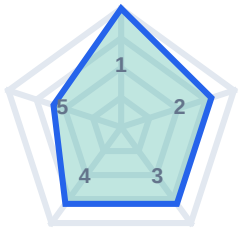
Overall

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	Leakage detection	10/10
2	Maps features to label timing	8/10
3	Risk of invalid model	8/10
4	Clear keep/remove list	8/10
5	Safer feature alternatives	6/10

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.

- 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: 19201 chars - Latency: 87s - 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: 1m 27s

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

Engineering counterpart
3:30 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
3:30 AM - Responded in 17s

Okay, here's a review of the candidate features for churn prediction, specifically focusing on potential leakage relative to the prediction time (first day of each month, 00:00 UTC), with recommendations for removal and safer alternatives. I'll organize this into categories: High Leakage, Potential Leakage, and Safe Features. I'll explain the reasoning, the leakage risk, and suggestions for resolution.

Understanding the Core Issue: Leakage

Feature leakage occurs when a feature contains information that would not be available at the prediction time, but inadvertently provides knowledge of the future (i.e., the target variable). In this case, it means the feature incorporates data from after the first day of each month, when the model is making its prediction. This leads to artificially inflated model performance during training/validation, but poor generalization to new, unseen data.

---

1. High Leakage - Remove Immediately

These features directly incorporate post-prediction time information and must be removed.

`account_status_is_churned`: Why it leaks: The very nature of this feature - indicating whether the account is churned - explicitly tells you the outcome you're trying to predict. The model would trivially learn to predict "churned" whenever this feature is 1. It's the ultimate label leakage. Recommendation: Remove completely. There's no safe alternative as this is the churn indicator itself.

`cancel_request_submitted_last_7d` & `cancel_request_submitted_last_30d`: Why it leaks: A cancellation request submitted after the prediction date (first of the month) provides direct insight into future churn. A customer submitting a request implies they are likely to churn, and including this information invalidates the prediction. Recommendation: Remove completely. Safer alternatives are discussed below in the "Potential Leakage" section.

2. Potential Leakage - Careful Review & Transformation Needed

These features may leak, depending on how they're calculated and stored. These require very careful consideration and potential transformation.

`days_until_contract_end`: Why it leaks: This is tricky. If the contract end date is updated after the first of the month (e.g., due to contract modification), then this feature includes information that wasn't available at prediction time. Even if the contract end date is only set when the contract is signed, if the model is deployed with a lag, this could cause leakage. Recommendation:

Option 1 (Strongly Preferred): Store the `days_until_contract_end` as of the previous month's first day. This ensures the data used is available at prediction time. This is the safest approach.

Option 2 (If Option 1 is not possible): Carefully audit when `days_until_contract_end` is updated. If it's never updated after the contract is initially signed and the model deployment isn't significantly lagged, it might be safe, but proceed with extreme caution and rigorous validation.

`nps_score_last_quarter`: Why it leaks: NPS scores are typically collected periodically (e.g., quarterly). If the NPS score for the current quarter isn't available on the first of the month, it leaks. This is very common and a significant issue. Consider if the score is collected before the prediction date, or if data collection is delayed. Recommendation:

Option 1 (Strongly Preferred): Use

Engineering counterpart
3:30 AM - Responded in 0s

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

Agent response
3:31 AM - Responded in 18s

Okay, let's break down my reasoning step-by-step for the feature leakage review, explicitly outlining the assumptions I made along the way. I'll structure this to mirror the categories I presented previously: High Leakage, Potential Leakage, and Safe Features.

I. High Leakage Features - The Logic & Assumptions

Feature: `account_status_is_churned`

Reasoning: This is the most straightforward. The definition of a "churned" account is that it has churned. Including this as a feature means the model is being told the answer directly, making prediction trivial. The purpose of a predictive model is to infer churn likelihood based on prior behavior, not to be told what's already happened.

Assumption: I assumed `account_status_is_churned` is a flag that is set when the account has already churned, not a predictive indicator. This is a very typical pattern. If, somehow, this flag was being set before churn actually occurred, the problem would be far more complex and require investigating the underlying data processes (which is outside the scope of this review).

Why it's definitive leakage: Because it directly reveals the target variable.

Feature: `cancel_request_submitted_last_7d` & `cancel_request_submitted_last_30d`

Reasoning: A cancellation request is an action taken with the intent to churn. Including the fact that a request was submitted within a timeframe after the prediction point means you are using information about a future event to predict that same event.

Assumption: I assumed these flags are set when the request is submitted, not before, and that the request processing/confirmation of churn happens after the prediction window. If a request is processed and the account is marked as churned immediately upon submission, there might be a less severe leakage issue (but still problematic), and we'd need to examine processing timelines. If the request triggers a churn event immediately, this becomes more of a "target variable" issue.

Why it's definitive leakage: Directly incorporates information about a future action taken because of an intended churn.

II. Potential Leakage Features - The Nuance & Risk

These required more careful consideration due to the possibility of leakage depending on data generation and update processes.

Feature: `days_until_contract_end`

Reasoning: The potential leakage hinges on when this value is calculated and updated. If the contract end date can be modified after the prediction date (first of the month), then the "days until end" reflects future changes.

Assumption 1 (Critical): I assumed that contract end dates are typically set at contract signing and rarely, if ever, changed after that date. This is a common practice, but not universally true. If contract amendments are frequent, this assumption is invalid.

Assumption 2: I assumed the model deployment doesn't have a significant lag. If the model is trained and updated, and then deployed with a delay of several weeks, then even a contract end date at signing could leak. The more lag, the more risk.

Why it's potentially leaky: Updates after the prediction time introduce knowledge of future events.

Feature: `nps_score_last_quarter`

Reasoning: NPS scores are usually collected at specific intervals (quarterly). If the scores for the current quarter are not available at the beginning of the month, including this feature

Engineering counterpart
3:31 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response
3:31 AM - Responded in 17s

Okay, you've got a solid feature leakage review, but before shipping this to production, here's a comprehensive checklist of verification steps I'd take, broken down into categories: Data Lineage & Definition, Temporal Validation, Model Performance Monitoring, and Governance & Documentation. I'll also include the level of effort (Low, Medium, High) required for each.

1. Data Lineage & Definition Verification (Medium - High Effort)

Detailed Data Dictionary Review: For every feature, go beyond just the name and definition provided. I'd want to see:
Source System: Where does the data originate? (e.g., CRM, Billing System, Support Platform)
Data Extraction Process: How is the data extracted from the source? (SQL query, API call, ETL job) - This helps identify potential points where biases or errors could be introduced.
Transformation Logic: Explicitly document every step of the feature engineering process. How is each feature calculated? What code is used? Who wrote it?
Update Frequency: When is the data refreshed? (Daily, Weekly, Monthly - is it in sync with the prediction window?)
Data Ownership: Who is responsible for the accuracy and consistency of the data?
Code Review: Thoroughly review the code that generates the features, paying close attention to:
Date/Time Handling: Are date/time calculations correct and consistent?
Filtering/Aggregation: Are any filters or aggregations being applied that might introduce bias or leakage? (e.g., are we accidentally including events after the prediction date?)
Null/Missing Value Handling: How are missing values handled? Are missing values themselves potentially predictive of churn?
Business Stakeholder Interviews: Talk to the people who understand the business processes that generate the data. Ask them:
"Can this feature's value ever change after the first of the month?"
"Are there any situations where this feature could reflect future events?"
"How is this feature used in other business decisions?" (This can reveal unintended uses or dependencies.)

2. Temporal Validation (High Effort - critical for this review!)

Time-Based Data Slicing: Take a representative sample of your training and validation data and explicitly slice it by the prediction time (first of each month).
Verify Data Consistency: For each feature, confirm that the values are consistent with what should be available before the first of the month. For example, for `days_until_contract_end`, verify that the contract end date used hasn't been updated after the first of the month.
Look for Temporal Anomalies: Are there any sudden jumps or shifts in feature values around the prediction time? This could indicate that data is being updated incorrectly.
Backtesting: Simulate production predictions using historical data.
Use data before the prediction date to generate features and make predictions.
Evaluate the accuracy of the predictions using actual churn outcomes.
Repeat this process over time to assess the model's stability and identify any degradation in performance.

3. Model Performance Monitoring (Medium Effort - ongoing after deployment)

Leakage Detection Metrics: Implement metrics to explicitly monitor for potential leakage after deployment. This is proactive.
Feature Distribution Shifts: Monitor for sudden shifts in the distribution of features around the prediction time. Significant changes could indicate data quality issues or leakage.

Engineering counterpart
3:31 AM - Responded in 0s

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

Agent response
3:31 AM - Responded in 14s

Okay, let's outline the top risks associated with deploying this churn model (especially given the leakage review) and define a robust rollback plan.

I. Top Risks (Prioritized)

Here's a breakdown of the highest risks, categorized by likelihood and impact, along with mitigation strategies planned before deployment.

1. High Likelihood, High Impact: False Negative Rates Increase Significantly.
Description: We've been aggressive in removing potentially leaky features. While this reduces the risk of overfitting, it could also remove valuable predictive signals, leading the model to underestimate churn risk and miss customers who will churn. This leads to missed intervention opportunities and increased churn.
Mitigation (Pre-Deployment):
Thorough Validation Set Evaluation: Heavily weight the validation set evaluation toward correctly identifying churned customers (prioritize recall/sensitivity, not just accuracy).
Champion/Challenger Testing: Run the new model alongside the existing model (or a baseline) in a parallel A/B test environment before full deployment. Closely monitor churn rates in each segment.
Scenario Testing: Simulate various churn scenarios (e.g., customers with contract renewals, reduced usage) and assess the model's predictions.
2. Medium Likelihood, Medium Impact: Model Performance Degradation Over Time.
Description: Even with careful verification, subtle data changes or shifts in customer behavior can cause the model's performance to degrade over time. Previously safe features might become leaky due to changes in data processing.
Mitigation (Pre-Deployment):
Robust Monitoring Framework (see previous response): Implement a comprehensive monitoring system that tracks feature distributions, model performance metrics, and business outcomes.
Automated Retraining: Schedule regular model retraining (e.g., monthly) on fresh data to adapt to changing patterns.
3. Low Likelihood, Medium Impact: Unexpected Feature Interactions & Biases.
Description: Even if individual features are not leaky, combinations of features or interactions can introduce biases that lead to unfair or inaccurate predictions for certain customer segments.
Mitigation (Pre-Deployment):
Fairness Auditing: Analyze model predictions across different demographic or customer segments to identify potential biases. Tools and libraries are available for this.
Explainable AI (XAI): Use techniques like SHAP values or LIME to understand why the model is making certain predictions and identify any unexpected feature contributions.
4. Low Likelihood, Low Impact: Data Quality Issues.
Description: Unexpected data errors or inconsistencies can creep into the data pipeline, impacting model performance.
Mitigation (Pre-Deployment): Implement data quality checks and validation rules at various stages of the pipeline.

II. Rollback Plan (Detailed & Tested)

The goal of the rollback plan is to revert to the previous state quickly and safely if the new model performs significantly worse than the baseline. This needs to be tested in a non-production environment.

1. Rollback Trigger Criteria: Define clear, quantifiable triggers that will initiate the rollback. Examples:
Significant Decline in Recall/Sensitivity: If the new model's recall drops by more than X% compared to the baseline model within Y days of deployment. (X and Y should be pre-defined and reasonable).
Increase in False Negative Rate: A statistically significant increase in the false negative rate (missing churners) compared to the baseline.
Champion/Challenger Test Failure: The champion model consistently underperforms the challenger model

Engineering counterpart
3:31 AM - Responded in 0s
Can you tighten this into concrete next steps with owners and timing?

Agent response
3:31 AM - Responded in 17s
Okay, let's solidify this into concrete next steps with owners and timelines. This assumes a relatively agile development & deployment process. I'll organize it into categories mirroring the previous risk assessment (Data Validation, Model Performance Monitoring, Fairness/Bias, and Rollback Readiness) and include estimated timelines. Please note: these are estimates and will need adjustment based on your specific team size, infrastructure, and complexity.

I. Data Validation & Lineage (Timeline: 1-2 Weeks)

Task	Description	Owner	Timeline	Dependencies
1.1 Detailed Data Dictionary Completion	Fully document source systems, transformation logic, update frequency, and data ownership for every feature.	Data Engineer/Data Governance	3 days	None
1.2 Code Review & Refinement	Review feature engineering code specifically for date/time handling, filtering, and aggregation logic. Address any identified issues.	Data Scientist/Data Engineer	5 days	1.1
1.3 Business Stakeholder Interviews	Conduct structured interviews with business stakeholders to validate data definitions and identify potential leakage risks.	Business Analyst/Data Scientist	2 days	1.1
1.4 Temporal Data Slicing & Validation	Slice historical data by the prediction time (first of the month) and verify feature values consistency.	Data Scientist	2 days	1.2

II. Model Performance Monitoring Setup (Timeline: 1-2 Weeks - Overlapping with Data Validation)

Task	Description	Owner	Timeline	Dependencies
2.1 Monitoring Dashboard Design	Design a dashboard displaying key performance metrics (recall, precision, F1-score, false negative rate), feature distributions, and business outcomes (churn rate).	Data Scientist/ML Engineer	3 days	1.4
2.2 Monitoring Implementation	Implement automated monitoring pipelines and alerts based on the dashboard design.	ML Engineer/DevOps	5 days	2.1
2.3 Champion/Challenger Test Setup	Set up an A/B testing framework to compare the new model against the existing model or a baseline.	ML Engineer	2 days	2.2

III. Fairness and Bias Assessment (Timeline: 0.5 - 1 Week - Can run concurrently)

Task	Description	Owner	Timeline	Dependencies
3.1 Segment Definition	Define relevant customer segments for fairness auditing (e.g., by contract type, industry, customer size).	Business Analyst/Data Scientist	1 day	None
3.2 Bias Detection Analysis	Run bias detection algorithms and analyze model predictions across the defined segments.	Data Scientist	3 days	3.1
3.3 Mitigation Strategy (if needed)	If bias is detected, develop and implement mitigation strategies (e.g., re-weighting, feature engineering).	Data Scientist	2-5 days (contingent)	3.2

IV. Rollback Readiness & Testing (Timeline: 0.5 - 1 Week - Final Stage)

Task	Description	Owner	Timeline	Dependencies
4.1 Rollback Plan Documentation	Document the complete rollback procedure, including steps for reverting the model, data pipelines, and infrastructure.	DevOps/ML Engineer	1 day	All prior tasks
4.2 Rollback				

Engineering counterpart
3:31 AM - Responded in 0s
What's the one thing you'd want a second pair of eyes on?

Agent response
3:32 AM - Responded in 4s
If I had to pick one thing I'd want a second pair of eyes on, it would be the temporal data slicing and validation (Task 1.4).

Here's why:

Subtlety of Leakage: Feature leakage, particularly subtle temporal leakage, is notoriously difficult to detect. It requires a very careful and methodical examination of the data, and it's easy to miss something when you're deeply involved in the process.
High Impact of Error: A failure to identify temporal leakage can invalidate the entire model and lead to significant business consequences.
Domain Expertise: A fresh perspective, especially from someone with strong domain knowledge of the data generation processes and business rules, can often spot inconsistencies or assumptions that the original analyst might have overlooked.

Essentially, having another person independently verify that the data used to train and evaluate the model truly reflects the information available at the prediction time is the most critical safeguard against deploying a leaky model. While all tasks are important, this one carries the highest risk of overlooking a subtle but catastrophic flaw.

Engineering counterpart
3:32 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.