Vantage RuntimeAI

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

7.2/10

Overall score

Run summary

Scenario: Feature Leakage Review
Model: openai/gpt-4.1-nano
Roles: Data Scientist (AI agent) <> Engineering counterpart (AI agent)
Duration: 33s
Actual run cost: $0.001740
Projected execution cost: $0.3828/mo (10/biz day × 22 days)
Report date: July 14, 2026 - 3:37 AM EST

Session: sample-lb-rankings-seed-b61ef8bc-f436-433c-b614-0403d17cab37

ds_feature_leakage_v1 - agent - openrouter - openai/gpt-4.1-nano - ended

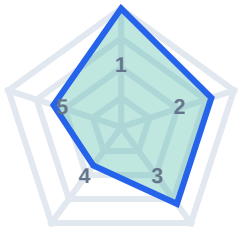
Overall

7.2/10

PASS - trust=high

Pass - score 7.2/10 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	4/10
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: Not met

TECHNICAL JUDGEMENT

Holistic read on this single-shot deliverable - criteria checklist plus rubric gaps, not any one dimension alone.

6/10

Telemetry

Run metrics

Mode: multi-turn - Agent msgs: 6 - Counterpart msgs: 7 - Output: 16648 chars - Latency: 33s - Criteria: 2/3

Score analysis

2/3 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. 2 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.

RUBRIC GAPS

- Clear keep/remove list (critical gap): Hard to follow or lacks concrete deliverable formatting.
- Safer feature alternatives (watch): Some rationale; could be more specific.

Signals

Flags leaking features: yes - Prediction-time framing: yes - Features to remove listed: no

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: 33s

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

Engineering counterpart
7:37 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
7:37 AM - Responded in 5s

Let's analyze each feature in relation to the prediction time (the start of each month) to identify potential leakage of the label or future information.

---

1. days_since_last_login
- Assessment: This is a measure of recent activity. If it aggregates login data up to the prediction date (the start of the month), it does not leak future information.
- Verdict: Safe (assuming data is only up to the prediction date).

---

2. support_tickets_last_30d
- Assessment: If this counts support tickets within the 30 days before the prediction date, it only includes past data, not future.
- Verdict: Safe (assuming proper data cut-off).

---

3. mrr_usd
- Assessment: MRR (monthly recurring revenue) at the start of the month reflects the current revenue, but if this is snapshotted after some churn activity has occurred, it may be contaminated with post-churn data.
- Key Point: If the MRR is measured at prediction time (the start of the month), it's safe.
- Verdict: Safe (assuming snapshot at prediction time).

---

4. contract_renewal_date
- Assessment: The contract renewal date is a fixed future date. It can leak if the model uses the fact that the renewal date is approaching, indicating potential churn.
- Concern: The mere presence of a renewal date isn't necessarily leakage, but if the information about renewal date is used to infer likelihood of churn, then it might leak.
- Recommendation: Use only the presence or absence, or whether renewal is upcoming after prediction time, not the exact date.

---

5. days_until_contract_end
- Assessment: Similar to renewal_date, if this measures days until contract ends at prediction time, it's safe.
- Concern: If this feature is calculated based on data after the prediction point, it leaks. For example, if the days until end are known after a customer has churned, it leaks.
- Recommendation: Ensure this is computed at the prediction date, not after.

---

6. cancel_request_submitted_last_7d
- Assessment: If the cancel request was submitted before the start of the month, it's only past data.
- Verdict: Safe, assuming data is only from before the prediction date.

---

7. cancel_request_submitted_last_30d
- Assessment: Same as above, but a longer window.

- Verdict: Safe, assuming data is up to the prediction date.

---

8. account_status_is_churned
- Assessment: If this flag indicates current account status at prediction time, it's safe.
- Concern: If this indicator is set after churn occurs (i.e., after the prediction date), then it leaks true label info.
- Recommendation: Use only the account status before or at the prediction date.

---

9. last_invoice_paid_amount
- Assessment: If this is the amount paid for the last invoice before the prediction date, it's non-leakage.
- Concern: If it reflects a payment after the prediction date (e.g., a payment made due to churn), then it leaks.
- Recommendation: Ensure it is only from invoices paid before the prediction date.

---

10. nps_score_last_quarter
- Assessment: Likely collected before or up to the prediction time, so it doesn't leak.
- Verdict: Safe if the NPS score is from data before prediction date.

Summary of leakage

Engineering counterpart
7:37 AM - Responded in 0s

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

Agent response
7:37 AM - Responded in 10s

Certainly! Here's a detailed step-by-step walkthrough of my reasoning process, including the assumptions I made:

---

1. Understanding the Prediction Context
- Prediction Time: The start of each month at 00:00 UTC.
- Target: Whether a customer will churn within 30 days after prediction time (i.e., within the month).

Assumption: The data used to generate features should only include information up to that prediction time, not future or post-prediction data.

---

2. General Principles for Leakage Assessment
- Leakage occurs when features contain information from the future relative to prediction point or directly reflect label-related data gathered after or as a consequence of the event we're trying to predict.
- Features reflecting the customer's status or behavior only up to the prediction date are generally safe.
- Features reflecting the customer's future state, or aggregated/processed data that includes post-prediction signals, are leakage.

---

3. Feature-by-Feature Analysis

a. days_since_last_login
- Reasoning: If this measures login activity up to the prediction date, it doesn't leak future information.
- Assumption: Data is cut off at prediction time; no login data beyond that point.

b. support_tickets_last_30d
- Reasoning: Counting support tickets within the 30 days before the prediction date is purely historical.
- Assumption: Data obtained up to the prediction date.

c. mrr_usd
- Reasoning: MRR at the start of the month reflects current revenue, not future or post-churn data.
- Assumption: Snapshot is taken at prediction time, before any churn occurs in that month.

d. contract_renewal_date
- Potential leakage: Knowing that a renewal date is approaching might hint at churn risk, but the date itself is fixed and exists regardless.
- Key: If this feature is used in the model, it could serve as a proxy for impending churn, which might be acceptable if the renewal date is publicly available or known before churn occurs.
- Assumption: The exact renewal date is known at or before prediction time and doesn't change based on future customer behavior.

e. days_until_contract_end
- Same reasoning as renewal_date. If it's a fixed, known value at prediction time, it's safe.
- Assumption: Calculated at the prediction date; no future knowledge.

f. cancel_request_submitted_last_7d / 30d
- Reasoning: If counts are based on past data before prediction, safe.
- Assumption: Data collection stops at prediction time; no post-prediction cancellations.

g. account_status_is_churned

- Potential leakage: If this reflects the current account status before churn, safe.
- But: If this status is updated after churn occurs (e.g., during data collection), it leaks label information.
- Assumption: Used data up to the prediction date; account status is frozen at or before that point.

h. last_invoice_paid_amount
- Reasoning: Payments before prediction time are safe; payments after could indicate churn or customer activity that occurs post prediction.
- Assumption: Only invoices paid before prediction are used.

i. nps_score_last_quarter
- Reasoning: NPS scores are usually collected periodically, and scores from before prediction are safe.
- Assumption: This score is from before the prediction date.

---

4. Key Assum

Engineering counterpart
7:37 AM - Responded in 0s
What would you verify before shipping this to production?

Agent response
7:37 AM - Responded in 5s
Before deploying this feature set to production, I would verify the following:

---

1. Data Cutoff and Timing Consistency
- Ensure temporal alignment: Confirm that all features are generated using data only up to the prediction date (the start of the month). No features should incorporate information from after that point.
- Verification Methods:
  - Check data extraction scripts to ensure proper data filtering.
  - Review data pipelines to confirm they cut off data at the prediction timestamp.

2. Feature Calculation Methodology
- Confirm that date-based features (like `contract_end_date`, `renewal_date`, `support_tickets`, `last_invoice_paid_amount`, etc.) are calculated at prediction time, not after.
- Verification:
  - Cross-validate sample feature values against raw data timestamps.
  - Use sample data where the timing is known to validate calculations.

3. Data Leakage in Data Pipeline
- Ensure that the data processing pipeline:
  - Is well-segmented into training and inference modes to prevent information from leaking from future data.
  - Has proper temporal splits in train/test data to simulate real-world prediction conditions.

4. Feature Stability and Replicability
- Test the features for stability:
  - Are features consistent over different data snapshots?
  - Do they reflect only past information?
- Verify that features don't rely on any future known events or labels.

5. Handling of Special Cases
- Missing data:
  - Confirm how missing or incomplete data is handled.
  - Ensure no assumptions that could introduce leakage or bias.

6. Documentation and Code Review
- Document assumptions about data timing and feature calculation.
- Have peers review code to catch potential leakage issues.

7. Monitoring After Deployment
- Set up validation checks to monitor feature distributions and model performance over time to detect unforeseen leakage or drift.

In summary:
The key verification step is to confirm that all features are truly historical relative to the prediction point, no future data is included, and the data pipeline enforces this temporal boundary rigorously before deployment.

Engineering counterpart
7:37 AM - Responded in 0s

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

Agent response
7:37 AM - Responded in 6s

Great questions. Addressing the key risks and having a clear rollback plan are crucial for a safe deployment.

Top Risks

1. Feature Leakage in Production
   - Description: If features inadvertently include data from after the prediction time (e.g., future support tickets, invoice payments, or account statuses), the model might appear highly accurate during validation but perform poorly on new, real-world data.
  - Impact: Overestimated model performance, poor generalization, and high operational risk.

2. Data Pipeline Errors
  - Description: Mistakes in data extraction or transformation scripts could lead to inconsistent or contaminated features.
  - Impact: Mismatch between training and production data, leading to model degradation.

3. Temporal Data Drift
  - Description: Changes in customer behavior or data collection processes over time might cause features to become stale or less predictive.
  - Impact: Reduced model accuracy over time.

4. Incorrect Assumptions About Data Timing
   - Description: Misunderstanding when certain features are computed (e.g., assuming support tickets are only past data, but they include future data).
  - Impact: Leakage and invalid model evaluation metrics.

5. Operational Failures
   - Description: Deployment issues, such as incorrect feature versions, environment mismatches, or failures in scheduled data refreshes.
  - Impact: Model operates on outdated or incorrect data, leading to unreliable predictions.

Rollback and Mitigation Plan

1. Pre-Deployment Checks
- Validate data pipeline: Run data quality checks, sample features, and verify timestamps match expectations.
- Conduct backtests: Compare model predictions using historical data with known labels to ensure no leakage.

2. Deployment Strategy
- Canary Deployment:
  - Deploy the model to a small subset of users or clients first.
  - Monitor performance metrics and data quality.
- Shadow Mode:
  - Run the new model in parallel with the existing system, without impacting end-users.
  - Compare predictions and performance.

3. Rollback Procedures
- Immediate Rollback:
  - If leakage or other issues are detected, revert to the previous production model/version.
- Data and Code Reversion:
  - Switch back to the last known good data pipeline and feature set.
- Monitoring and Alerting:

- Continue monitoring key metrics and set alerts for anomalies or performance drops.

4. Communication and Documentation
- Inform stakeholders promptly about issues.
- Document the reason for rollback and lessons learned to improve future deployments.

Summary
- The top risks center around feature leakage, data inconsistencies, and operational failures.
- A robust, staged deployment with testing, monitoring, and ready rollback procedures minimizes potential negative impacts.
- Having pre-validated data pipelines and version control ensures quick recovery if issues arise.

Would you like a detailed checklist or specific technical steps for any part of this plan?

Engineering counterpart
7:37 AM - Responded in 0s
Can you tighten this into concrete next steps with owners and timing?

Agent response
7:37 AM - Responded in 4s
Certainly! Here's a structured, concrete next steps plan with clear owners and timing to ensure a safe deployment and quick rollback if needed:

Next Steps Plan for Deployment

Step	Description	Owner	Timing	Notes
1. Data Validation & Leak Check	Verify that all feature extraction scripts only use data up to the prediction timestamp. Sample and compare features against raw data for several customers.	Data Engineer / Data Scientist	Within 2 days	Use a small sample to ensure no future data leaks.
2. Pipeline Review & Testing	Review data pipelines for temporal consistency. Run end-to-end tests, including data refreshes and feature calculation steps.	Data Engineer	Within 3 days	Ensure pipeline enforces data cutoffs and versioning.
3. Model Performance Validation	Backtest the model on historical data, simulating real prediction timing to confirm no leakage.	Data Scientist	Within 1 day	Use holdout sets with known outcomes.
4. Deploy to Canary Environment	Deploy the model and features to a small subset of users or a sandbox environment for monitoring.	ML Engineer / DevOps	Day 4	Set up monitoring dashboards and alerts.
5. Monitor & Validate in Production (Canary)	Track model predictions, feature distributions, and performance metrics closely for at least 2-3 days.	Data Scientist / ML Ops	Days 5-7	Look for anomalies indicating leakage or data issues.
6. Decision Point: Go / Rollback	Review monitoring results. If issues are detected, prepare to rollback immediately.	Stakeholders / Project Lead	Day 8	Decide based on observed performance and data integrity.
7. Full Deployment / Rollback Execution	If all is well, deploy the model to full production. If issues are identified, revert to previous stable version.	DevOps / ML Engineer	Day 9	Ensure version control and quick rollback scripts are ready.
8. Post-Deployment Monitoring	Continue monitoring for data drift, performance, and leakage signals.	Data Scientist / ML Ops	Ongoing	Set alerts for significant deviations.

Roles Clarification
- Data Engineer: Responsible for data pipeline review, extraction, and validation.
- Data Scientist: Responsible for model validation, backtesting, and monitoring.
- ML Engineer / DevOps: Responsible for deployment, environment setup, and rollback execution.
- Stakeholders / Project Lead: Responsible for decision-making based on progress.

Summary
- Complete data and pipeline validation within 2-3 days.
- Conduct controlled deployment with thorough monitoring over 3 days.
- Decide on full deployment or rollback by day 8.
- Ensure readiness for quick rollback with versioning and scripts.

Would you like me to prepare email templates for updates or more detailed technical documentation?

Engineering counterpart
7:37 AM - Responded in 0s
What's the one thing you'd want a second pair of eyes on?

Agent response
7:37 AM - Responded in 2s
The one thing I would prioritize for a second pair of eyes is the correctness and timing of feature calculations, specifically ensuring that all features are derived solely from data available up to the prediction date and do not incorporate any information from after the prediction point or future events (like post-prediction churn, invoice payments, or account status updates).

Why?
Because even a small oversight-such as mistakenly including support tickets submitted after the prediction date or using a renewal date that's only known after churn-can lead to significant leakage, inflating model performance during validation but ultimately causing poor real-world results.

What to review:
- The data extraction scripts and code snippets for each feature.
- The timestamps and data filtering logic used during feature engineering.
- Sample data points to double-check that feature values align with the intended temporal window.

In essence:
A second pair of eyes on the feature calculation methodology ensures the features are strictly historical relative to the prediction time, preventing leakage and ensuring model validity when deployed in production.

Engineering counterpart
7:37 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.