



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: openai/gpt-4.1-mini

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

Duration: 1m 18s

Actual run cost: \$0.006960

Projected execution cost: \$1.53/mo (10/biz day × 22 days)

Report date: July 14, 2026 - 3:37 AM EST

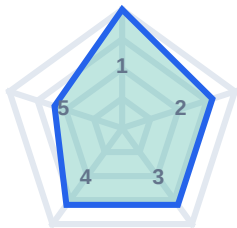
Overall

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	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: 19900 chars - Latency: 78s - 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 18s

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

Engineering counterpart

7:35 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

7:36 AM - Responded in 17s

Here is the leakage assessment for each candidate feature given the prediction task:

1. days_since_last_login

- Leakage risk? No
- Why: This feature reflects user activity up to the prediction time (start of month), so it uses only historical data.
- Recommendation: Keep.

2. support_tickets_last_30d

- Leakage risk? Potential risk if "last 30 days" includes post-prediction period
- Why: At prediction time (day 1 of the month), "last 30 days" should mean days -30 to 0, i.e. the month prior. If this is properly enforced, no leakage. But if computed dynamically including days after prediction time (e.g., days 1-30 of current month), it leaks future info.
- Recommendation: Ensure the 30-day window ends at prediction time. If so, keep; else remove or re-compute safely.

3. mrr_usd (Monthly Recurring Revenue)

- Leakage risk? No
- Why: It is a snapshot at prediction time showing current revenue; no future info involved.
- Recommendation: Keep.

4. contract_renewal_date

- Leakage risk? Potentially risky
- Why: If the renewal date is known and fixed from the contract start, this is fine. But if this date is updated post-churn or after cancellation (e.g., adjusted after customer signals churn), it leaks info. Also, if renewal is after prediction + 30d, this is safe. However, if it sometimes moves forward due to events in the future, it leaks.
- Recommendation: Verify if renewal date is pre-known and static. If yes, keep. Else, remove.

5. days_until_contract_end

- Leakage risk? Same considerations as contract_renewal_date
- Why: This is derived from contract_renewal_date minus prediction time. If the contract end date can adjust after churn or cancellation signals, it leaks.
- Recommendation: Same as above-keep only if fixed ahead of time.

6. cancel_request_submitted_last_7d

- Leakage risk? Yes (leaks label/post-churn info)
- Why: A cancel request submitted in the 7 days before prediction time is valid. But if this window includes days after prediction (e.g., 7 days into future), it leaks. More importantly, if the cancel request occurs within the prediction window (0-30 days after prediction), it is directly related to the label (churn within 30 days). Using this means you already know the churn event happened or is imminent, which leaks label info.
- Recommendation: Remove or only use cancel requests up to prediction time. But even then, a cancel request immediately before prediction is a strong proxy for churn and may inflate performance artificially.

7. cancel_request_submitted_last_30d

- Leakage risk? Yes, same reasoning as last_7d but more so
- Why: If the 30 days includes any time after prediction, it leaks. Even if it ends at prediction time, a recent cancel request is highly predictive and close to label, thus "soft label leakage."
- Recommendation: Remove or carefully truncate window ending at prediction time. Consider excluding to avoid label leakage.

8. account_status_is_churned

- Leakage risk? Yes, direct leakage
- Why: If account_status is "churned" at prediction time, the customer already churned. The label is about churn within 30 days after prediction

Engineering counterpart
7:36 AM - Responded in 0s

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

Agent response

7:36 AM - Responded in 15s

Certainly! Here's a detailed walkthrough of my reasoning process, including assumptions made at each step.

Step 1: Understand the prediction task and timeline

- Goal: Predict whether a B2B SaaS customer will churn within 30 days starting from the first day of each calendar month (00:00 UTC).
- Prediction time: The start of the month (e.g., March 1, 00:00 UTC).
- Label: `churned\_within\_30d` means customer churns any time from prediction time up to 30 days later.
- Key principle: Features must only use data available up to the prediction time (exclusive of the future 30 days) to avoid leakage.

Step 2: Analyze each feature against the timeline

For each feature, I asked:

- Does it use data from after prediction time?
- Could it be influenced by events that happen during or after the 30-day window?
- Could it directly or indirectly encode information about the churn event or cancellation?

Step 3: Feature-by-feature reasoning with assumptions

Feature: days_since_last_login

- Assumption: This counts days from the last login date up to prediction time.
- Since last login must be before or at prediction time, no future information is used.
- Conclusion: Safe, no leakage.

Feature: support_tickets_last_30d

- Assumption: The "last 30 days" refers to the 30 days before prediction time (e.g., Feb 1 to Feb 28 if predicting on Mar 1).
- Risk is if "last 30 days" is computed incorrectly including future days (e.g., Mar 1 - Mar 30).
- Conclusion: Safe if window ends at prediction time; otherwise, leaks future info.
- Recommendation: Confirm window definition; fix as needed.

Feature: mrr_usd

- Assumption: MRR is a snapshot at prediction time reflecting current revenue status.
- No forward-looking or label-dependent data.
- Conclusion: Safe.

Feature: contract_renewal_date

- Assumption: Contract renewal dates are predetermined and fixed at contract signing.
- If renewal date is adjusted after churn or cancellation signals (e.g., postponed or canceled), it leaks future knowledge.
- Some companies update renewal dates dynamically (e.g., after re-negotiations).
- Conclusion: Conditional. Safe if renewal date is immutable and known at prediction time; otherwise, potential leakage.

Feature: days_until_contract_end

- Same assumptions as contract_renewal_date, as it is derived from it.
- Conclusion: Same as above.

Feature: cancel_request_submitted_last_7d

- Assumption: Cancel request timestamps are known accurately.
- If "last 7 days" includes any time after prediction, it leaks.
- Even if the window ends at prediction time, cancel requests close to prediction are strong signals of imminent churn.
- Using cancel requests near prediction time is almost like using label proxies.
- Conclusion: Likely label leakage or "soft leakage".
- Recommendation: Remove or only use cancel requests from a sufficiently large buffer period before prediction (e.g., >30 days ago) to avoid leakage.

Feature: cancel_request_submitted_last_30d

- Same logic as last_7d but stronger leakage due to longer window.
- Conclusion: Same as above, even stronger reason to remove.

Feature: account_status_is_churned

- Assumption: This status is updated immediately when churn occurs.
- Using this at prediction time means the customer already churned.
- Since label is churn within next 30 days, having churned customers at prediction time means label is already 1, causing

Engineering counterpart
7:36 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

7:36 AM - Responded in 12s

Before deploying this churn model to production, I would verify the following to ensure no leakage, robustness, and operational stability:

1. Feature Engineering Time Consistency

- Confirm all time windows are strictly historical:

Verify that features like `support\_tickets\_last\_30d`, `cancel\_request\_submitted\_last\_7d/30d`, `days\_since\_last\_login`, etc. use only data from before the prediction cutoff timestamp (start of month).

- Validate event timestamp alignment:

Check data pipelines to confirm no future-dated or misaligned timestamps creep into features.

2. Feature Stability and Definitions

- Contract-related features:

Ensure `contract\_renewal\_date` and `days\_until\_contract\_end` are fixed at contract inception or last known update date, and not updated retrospectively based on churn or cancellations.

- Cancel request features:

Decide whether to exclude or modify these features to avoid soft leakage. If used, confirm cancel requests do not overlap with label period.

3. Label Definition and Data Leakage

- Verify label correctness:

Confirm churn label `churned\_within\_30d` is calculated based on events strictly after prediction time.

- Check for overlap:

Ensure no features are unintentionally exposing parts of the label (e.g., `account\_status\_is\_churned` at prediction time should be false for all samples).

4. Data Sampling and Representativeness

- Confirm training data cutoff:

The training dataset should have no data from after each prediction time per sample.

- Test on realistic production-like data:

Validate model on holdout or future data simulation to detect leakage effects.

5. Performance and Explainability Checks

- Model explainability:

Feature importance/SHAP values should not be dominated by features that are borderline leakage proxies (e.g., cancel requests right before prediction).

- Sanity checks:

Test if removing suspect features causes performance drop; large drops may indicate leakage.

6. Monitoring and Alerts

- Setup monitoring to detect data drift or leakage:
For example, monitor feature distributions over time, especially for those features with leakage risk.
- Alert on label leakage patterns:
Unexpectedly high model performance or sudden changes in feature importances might indicate leakage slipping in.

7. Documentation and Collaboration

- Document assumptions and definitions for feature engineering team.
- Clarify with product or domain experts:
Confirm business rules around contract dates, churn signals, and cancel requests.

Summary: Key verification steps

| Verification | Purpose |
|-----------------------------------|--|
| Time window correctness check | No future data used in features |
| Contract dates immutability check | Avoid leakage from updated contracts |
| Cancel request feature audit | Avoid soft label leakage |
| Label correctness check | Ensure label matches prediction window |
| Data sampling consistency | No data leakage in train/test split |
| Model explanation review | Detect proxy features for churn |
| Monitoring setup | Catch leakage or drift in production |

Ensuring these steps will help prevent leakage, maintain model generalization, and build trust in production predictions.

Engineering counterpart
7:36 AM - Responded in 0s

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

Agent response

7:36 AM - Responded in 11s

Top Risks Before and After Production Deployment

| Risk | Explanation | Impact |
|--|---|---|
| 1. Feature leakage via cancel requests or account status | Using `cancel_request_submitted_last_7d/30d` or `account_status_is_churned` may reveal future churn events or the label itself. | Artificially high model performance, poor generalization, misleading insights, potential business misactions. |
| 2. Incorrectly defined time windows in usage or support data | Features like `support_tickets_last_30d` or `days_since_last_login` including data after prediction time. | Subtle leakage causing over-optimistic predictions; hard to detect until production. |
| 3. Contract date updates post-churn | Dynamic changes to `contract_renewal_date` or `days_until_contract_end` after churn or cancelation signals. | Indirect leakage; model learns future contract modifications linked to churn. |
| 4. Data pipeline bugs that introduce future data | Wrong joins, timestamp misalignments, or backfills create leakage. | Model performance collapse or unexplained behavior in production. |
| 5. Label mismatch or delayed updates | Label data not synchronized with prediction time; e.g., churn status updated late or early. | Model trained on incorrect labels; poor real-world performance. |

Rollback Plan if Model Fails or Leakage Detected

1. Immediate rollback to previous stable model/version
 - Revert to last production model known to perform reliably without leakage.
 - Switch off automatic decisions based on new model outputs if in an automated pipeline.
2. Freeze feature set and data pipeline versions
 - Lock feature computation at last known-good state to avoid further data contamination.
 - Perform root cause analysis on new features suspected of leakage.
3. Conduct thorough post-mortem and debugging
 - Audit feature engineering code and data windows.
 - Review data snapshots to confirm no post-prediction info slipped in.
 - Cross-check all timestamps and event logs with domain experts.
4. Retrain a safe baseline model
 - Use only features confirmed safe (e.g., `days\_since\_last\_login`, `mrr\_usd`) excluding high-risk features.
 - Evaluate on proper historical data splits.
5. Deploy monitoring with alerting for leakage signs
 - Monitor input feature distributions and label statistics for drift.
 - Add checks for impossible feature values or sudden jumps in feature importance.
6. Implement a more conservative feature gating policy
 - Temporarily disallow features close to label or future event timestamps until proven safe.
7. Communicate with business stakeholders
 - Notify downstream users of the rollback and confidence issues.
 - Adjust expectations and decision processes accordingly.

Summary of mitigation philosophy

- Fail fast and safe: Roll back immediately to stable known-good state on any sign of leakage or performance drop.
- Isolate and analyze: Use rigorous data audits and feature validation to find root cause.
- Iterate conservatively: Reintroduce risky features only after comprehensive validation, ideally after offline simulations and shadow testing.
- Maintain transparency: Keep stakeholders informed to avoid misinterpretation of churn predictions.

This plan balances risk control with continuous improvement, minimizing business impact from leakage or model failure.

Engineering counterpart
7:36 AM - Responded in 0s

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

Agent response

7:37 AM - Responded in 14s

Certainly! Here is a concrete, actionable next-step plan with owners and suggested timing to address the top risks and prepare a rollback plan before shipping:

Concrete Next Steps for Leakage Risk Mitigation & Rollback Preparation

| Step | Action | Owner | Timing | Notes |
|------|--|---|-------------------------------------|---|
| 1 | Audit feature time windows and definitions: Verify all features use only data up to prediction time (especially cancel requests, support tickets, contract dates). | Data Science + Feature Engineering Lead | 2 business days | Review code, SQL queries, ETL pipelines. |
| 2 | Confirm immutability of contract dates: Validate contract_renewal_date and days_until_contract_end are fixed and not updated post-churn. | Product Owner + Data Engineering | 2 business days | Coordinate with legal/contracts team if needed. |
| 3 | Remove or modify cancel request features: Either exclude these features from current model iteration or restrict to a historical window ending well before prediction time (e.g., >30 days). | Data Science Team | 3 days | Prioritize safe removal if uncertainty remains. |
| 4 | Validate label correctness and alignment: Ensure churn labels are correctly assigned based on events after prediction time with no leakage. | Data Science + Labeling/BI Team | 2 days | Cross-check with business definitions. |
| 5 | Implement automated data/time consistency checks: Add unit/integration tests to pipelines verifying no future data inclusion. | Data Engineering + ML Ops | 1 week | Automate safeguards for all production feature builds. |
| 6 | Run retrospective model training and validation: Train a version without risky features, compare performance to suspect-feature-included model to detect leakage impact. | Data Science Team | 3 business days | Use time-split validation identical to production flow. |
| 7 | Develop rollback deployment plan and checklist: Document precise steps to revert to last known-good model and freeze features/pipelines. | ML Ops + Data Science | 1 week | Prepare scripts and communication templates. |
| 8 | Set up monitoring and alerting: Create dashboards to track feature distributions, model performance, and anomalous changes in feature importances post-deployment. | ML Ops + Data Science | 1 week | Include early-warning triggers for leakage or drift. |
| 9 | Communicate with stakeholders: Inform business, product, and support teams of risks, rollback plan, and expected timelines. | Product Manager + Data Science Lead | Immediately (within 1 day) | Align expectations and prepare contingency plans. |
| 10 | Schedule post-deployment review: Plan a checkpoint (e.g., 2 weeks after deployment) to review model performance and any alerts. | Data Science Lead + ML Ops | Upon deployment + 2 weeks follow-up | Capture lessons learned and adjust processes. |

Summary:

- Immediate (1-3 days): Audit features, verify label, communicate risks, remove risky features if needed.
- Short-term (1 week): Implement pipeline checks, validate contract date assumptions, run comparative training experiments.
- Mid-term (1-2 weeks): Setup monitoring & alerting, develop rollback scripts, plan post-deployment review.

This plan assigns clear ownership, measurable timing, and prioritizes mitigating leakage risk while maintaining agility for deployment decisions.

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 8s

The one thing I'd want a second pair of eyes on is the precise time window definitions and implementations for "cancel_request_submitted_last_7d/30d" and "support_tickets_last_30d" features, including how these windows are computed in the data pipelines relative to prediction time.

Why this?

- These features carry the highest risk of subtle leakage because they are tightly coupled to churn events and can easily include future data if timestamps or window boundaries are misaligned.
- Even small off-by-one errors or timezone issues can cause features to incorporate post-prediction events, leaking label or future behavior.
- The decision whether to keep, truncate, or exclude these features hinges on correctly understanding and validating these windows.
- It is often difficult for a single person to spot subtle bugs in time-window logic, especially when multiple systems (databases, ETL jobs, ML feature stores) are involved.

What to ask the reviewer to check:

- Confirm that all timestamps used to filter these features are strictly `< prediction\_time` with no exceptions.
- Verify no mixing of timezone or daylight savings effects that shift the window boundaries.
- Review the SQL/feature code for any usage of `<=` vs `<` that could mistakenly include partial future days.
- Validate actual feature values on historical data slices to detect anomalies or impossible values (e.g., cancel requests after prediction date).
- Check alignment between business definitions of "last 7/30 days" and technical implementations.

This review will substantially reduce the risk of label leakage and improve confidence in the features feeding the churn model.

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.