



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: deepseek/deepseek-chat

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

Duration: 2m 39s

Actual run cost: \$0.004477

Projected execution cost: \$0.9849/mo (10/biz day × 22 days)

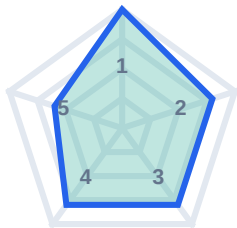
Report date: July 13, 2026 - 11:31 PM EST

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: 17095 chars - Latency: 159s - 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: 2m 39s

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

Engineering counterpart

3:28 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

3:28 AM - Responded in 18s

Leakage Analysis & Recommendations

1. Leaky Features (Remove Immediately)

These features either directly encode the label or use post-prediction-time information:

| Feature | Reason for Leakage |
|-------------------------------------|--|
| `cancel_request_submitted_last_7d` | If a cancel request was submitted in the last 7 days, the customer is almost certainly about to churn. This directly leaks the label. |
| `cancel_request_submitted_last_30d` | Similar to above-if a cancel request exists in the last 30 days, it likely means the customer will churn within the next 30 days (our prediction window). |
| `account_status_is_churned` | This is a direct label leak-if the account is already marked as churned, the prediction is meaningless. |
| `days_until_contract_end` | If the contract ends within the next 30 days, the customer is highly likely to churn (unless they renew). This encodes future information relative to prediction time. |

2. Potentially Leaky Features (Evaluate Carefully)

These features may introduce leakage depending on data availability timing:

| Feature | Risk | Safer Alternative |
|----------------------------|--|---|
| `days_since_last_login` | If computed at prediction time, it may include logins that happened after the prediction cutoff (leaking future behavior). | Use `days_since_last_login_as_of_previous_month_end` (lagged by 1 day). |
| `support_tickets_last_30d` | If computed at prediction time, it could include tickets raised after the prediction cutoff. | Use `support_tickets_previous_30d` (strictly prior to prediction time). |
| `last_invoice_paid_amount` | If the last invoice was generated after prediction time, it leaks future payment status. | Use `last_invoice_paid_amount_as_of_previous_month`. |

3. Safe Features (No Leakage)

These features are safe if computed strictly before prediction time:

- `mrr\_usd` (assuming it's the trailing MRR as of the previous month)
- `contract\_renewal\_date` (as long as it's a fixed contractual date, not a future renewal event)
- `nps\_score\_last\_quarter` (if the quarter ended before prediction time)

Final Recommendations

Remove these features immediately due to leakage:

- `cancel\_request\_submitted\_last\_7d`
- `cancel\_request\_submitted\_last\_30d`
- `account\_status\_is\_churned`
- `days\_until\_contract\_end`

Modify these features to avoid leakage:

- Replace `days\_since\_last\_login` with a lagged version (e.g., `days\_since\_last\_login\_as\_of\_previous\_month\_end`).
- Replace `support\_tickets\_last\_30d` with `support\_tickets\_previous\_30d` (strictly prior to prediction time).
- Replace `last\_invoice\_paid\_amount` with a lagged version.

Safe to keep (if properly time-bound):

- `mrr\_usd`
- `contract\_renewal\_date`
- `nps\_score\_last\_quarter`

This ensures the model only uses information available at prediction time and avoids lookahead bias.

Engineering counterpart

3:28 AM - Responded in 0s

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

Agent response

3:29 AM - Responded in 18s

Step-by-Step Reasoning for Leakage Detection

1. Understanding the Prediction Task

- Goal: Predict if a B2B SaaS customer will churn within the next 30 days.
- Prediction Time: First day of each month (00:00 UTC).
- Label Definition: `churned\_within\_30d = 1` if the customer churns between prediction day (Day 0) and Day 30.

2. Key Principle: No Future Information

- Strict Rule: Features must only use data available before or at prediction time (Day 0).
- Leakage Occurs If:
 - A feature directly encodes the label (e.g., `account\_status\_is\_churned`).
 - A feature depends on events after prediction time (e.g., `cancel\_request\_submitted\_last\_7d` could include requests submitted after Day 0).

3. Evaluating Each Feature

? High-Risk Leakage (Remove)

| Feature | Leakage Reason | Alternative? |

|-----|-----|-----|

| `cancel\_request\_submitted\_last\_7d` | If a cancel request was submitted in the last 7 days (Day -7 to Day 0), the customer is highly likely to churn within the next 30 days. This directly reveals the label. | No safe alternative-this is a proxy for churn. |

| `cancel\_request\_submitted\_last\_30d` | Similar to above-if a cancel request exists in the last 30 days (Day -30 to Day 0), the customer is likely to churn soon. | No safe alternative-still leaks future intent. |

| `account\_status\_is\_churned` | If the account is already marked as "churned," the label is already true. This is pure label leakage. | Remove entirely. |

| `days\_until\_contract\_end` | If the contract ends in <30 days (e.g., `days\_until\_contract\_end = 15`), the customer is likely to churn unless they renew. This encodes future churn risk post-prediction. | Safer: Use `contract\_renewal\_date` (fixed date) instead. |

?? Potentially Leaky (Require Time-Bound Fixes)

| Feature | Risk | Fix |

|-----|-----|-----|

| `days\_since\_last\_login` | If computed at prediction time (Day 0), it could include logins after the prediction cutoff (e.g., if data pipeline runs at 00:00 UTC but logins happen later that day). | Fix: Use `days\_since\_last\_login\_as\_of\_previous\_month\_end` (lagged by 1 day). |

| `support\_tickets\_last\_30d` | If calculated at prediction time, it might include tickets raised after Day 0 (e.g., tickets from Day 1 to Day 30). | Fix: Use `support\_tickets\_previous\_30d` (strictly Day -30 to Day -1). |

| `last\_invoice\_paid\_amount` | If the "last invoice" includes payments after prediction time (e.g., an invoice paid on Day 5), it leaks future payment status. | Fix: Use `last\_invoice\_paid\_amount\_as\_of\_previous\_month`. |

? Safe Features (If Properly Time-Bound)

| Feature | Why Safe? |

|-----|-----|

| `mrr\_usd` | Assuming it's trailing MRR (e.g., as of Day -1)

Engineering counterpart
3:29 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

3:29 AM - Responded in 23s

Before shipping this churn model to production, I would rigorously verify the following critical aspects to ensure reliability, fairness, and robustness:

1. Data Pipeline Validation

? Ensure features are computed strictly at prediction time (no future leakage):

- Confirm that all features (e.g., `support\_tickets\_previous\_30d`, `days\_since\_last\_login\_as\_of\_previous\_month\_end`) are lagged correctly and never include post-prediction-time data.
- Test: Run backtests with historical snapshots to validate no data from the future leaks in.

? Check for missing/imputed data:

- How are missing values handled? (e.g., `last\_invoice\_paid\_amount` for new customers)
- Test: Simulate edge cases (e.g., new signups, inactive users) to ensure robustness.

2. Model Performance & Stability

? Temporal validation (avoid time-based overfitting):

- Train on older data (e.g., Jan-Jun), validate on recent data (e.g., Jul-Dec).
- Ensure AUC/recall doesn't degrade in the holdout period.

? Segment-wise performance:

- Does the model perform equally well for:
 - High-MRR vs. low-MRR customers?
 - Short-tenure vs. long-tenure customers?
- Test: Stratified evaluation by key segments.

? Calibration check:

- Do predicted probabilities match actual churn rates? (e.g., if the model predicts 10% churn risk, does ~10% of that group actually churn?)
- Test: Use reliability curves or binning analysis.

3. Business Alignment

? Actionable thresholds:

- Does the chosen probability threshold (e.g., "flag if risk > 20%") align with retention costs?
- Test: Run a cost-benefit analysis (e.g., cost of intervention vs. saved MRR).

? Feature importance sanity checks:

- Do the top features make sense? (e.g., `days\_since\_last\_login` should negatively correlate with churn).
- Test: SHAP values or permutation importance.

4. Monitoring & Maintenance Readiness

? Set up drift detection:

- Monitor:
 - Feature distributions (e.g., sudden drop in `support\_tickets`).
 - Model performance decay (e.g., AUC drop over time).
- Tooling: Use tools like Evidently, Arize, or custom alerts.

? Document failure modes:

- What happens if:
 - Payment data is delayed?
 - A new product feature reduces logins (but not churn)?
- Test: Chaos engineering (simulate data outages).

5. Ethical & Compliance Checks

? Bias/fairness audit:

- Does the model disproportionately flag certain customer segments? (e.g., by geography, industry)
- Test: Disparate impact analysis (e.g., demographic parity ratios).

? GDPR/legal compliance:

- Are all features legally permissible? (e.g., using NPS data may require consent.)
- Test: Review with legal/privacy teams.

6. Stakeholder Buy-In

? Explainability:

- Can the model provide reasons (e.g., "high risk due to low login activity") for retention teams?
- Test: Generate sample explanations for top predictions.

? Pilot testing:

- Run a small-scale live test (e.g., 5

Engineering counterpart
3:29 AM - Responded in 0s

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

Agent response

3:30 AM - Responded in 1m

Top Risks When Deploying the Churn Model

1. Model Performance Drops in Production

- Risk: The model performs well in testing but degrades in real-world use due to:
 - Data drift (e.g., sudden changes in user behavior).
 - Concept drift (e.g., churn patterns shift post-launch).
 - Pipeline errors (e.g., incorrect feature computation).
- Mitigation:
 - Deploy in shadow mode (run predictions alongside old system without acting on them).
 - Use A/B testing (compare against a control group before full rollout).

2. False Positives Harm Customer Relationships

- Risk: Over-aggressive churn predictions lead to:
 - Unnecessary retention offers (costly).
 - Annoying outreach (e.g., excessive calls to loyal customers).
- Mitigation:
 - Start with high-confidence thresholds (e.g., only act on >30% predicted churn risk).
 - Implement a human review step for borderline cases.

3. False Negatives Miss At-Risk Customers

- Risk: The model fails to flag real churn risks, leading to preventable revenue loss.
- Mitigation:
 - Monitor recall (percentage of actual churners caught) closely.
 - Set up fallback rules (e.g., if model confidence is low, use a simple heuristic like "no login in 14 days").

4. Data Pipeline Failures

- Risk: Missing or corrupted data (e.g., login events not logged) breaks feature computation.
- Mitigation:
 - Validate inputs (e.g., reject predictions if >10% of features are missing).
 - Log all feature values for debugging.

5. Bias Against Certain Customer Segments

- Risk: The model unfairly targets/ignores groups (e.g., small businesses vs. enterprises).
- Mitigation:
 - Continuously monitor fairness metrics (e.g., churn prediction rates by segment).
 - Retrain with balanced sampling if disparities emerge.

Rollback Plan (If Model Fails)

1. Immediate Rollback Triggers

- Performance drop: If AUC falls below [X] or recall below [Y] for [Z] days.
- Customer complaints: Spike in support tickets about retention outreach.
- Data issues: Critical features fail to compute for >5% of predictions.

2. Rollback Steps

1. Switch to fallback model:
 - Revert to a simpler model (e.g., rule-based: "churn risk = 1 if no login in 30 days").
 - Or disable automation and default to human judgment.
2. Root-cause analysis:
 - Check for:
 - Data pipeline errors.

- Shifts in user behavior.
- Model staleness.

3. Communicate transparently:

- Inform stakeholders (e.g., "Model paused due to unexpected performance drift").

3. Post-Rollback Actions

- Fix and retrain: Address the issue (e.g., adjust features, retrain on newer data).
- Gradual re-release: Test fixes in staging before redeploying.

Key Takeaways

- Test rigorously before launch (shadow mode, A/B tests).
- Monitor aggressively (performance, fairness, data quality).
- Have a rollback button ready (fallback model + clear triggers).

This ensures the model adds

Engineering counterpart
3:30 AM - Responded in 0s

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

Agent response

3:30 AM - Responded in 23s

Here's a tight, actionable rollout plan with owners, timelines, and escalation paths:

Phase 1: Pre-Deployment Checks (Week 1-2)

| Task | Owner | Timing | Success Criteria |

|-----|-----|-----|-----|

| Validate feature pipelines (no future leakage) | Data Eng | Week 1 | Backtest confirms no leakage in historical snapshots |

| Finalize model monitoring (AUC, recall, drift) | ML Eng | Week 1 | Dashboards live in Grafana/DataDog |

| Legal review of features (GDPR/compliance) | Legal | Week 2 | Signed off for production use |

| Shadow mode deployment (log predictions) | DevOps | Week 2 | Predictions generated for 100% of users, no actions taken |

Phase 2: Soft Launch (Week 3-4)

| Task | Owner | Timing | Success Criteria |

|-----|-----|-----|-----|

| A/B test: 10% traffic to model-driven retention | Growth | Week 3 | No increase in churn or support tickets |

| Human review for top 10% risk scores | CX Team | Week 3-4 | False positive rate <5% in reviewed cohort |

| Daily performance checks (AUC >0.85) | Data Sci | Daily | Alert if metric drops for 2+ days |

Rollback Triggers:

- AUC drops below 0.8 for 3 days -> Revert to rule-based fallback (owned by DevOps).
- Customer complaints spike (>5% increase) -> Pause automated outreach (owned by Growth).

Phase 3: Full Launch (Week 5+)

| Task | Owner | Timing | Success Criteria |

|-----|-----|-----|-----|

| 100% rollout of model-driven retention | Growth | Week 5 | Churn rate decreases MoM |

| Weekly fairness audits (by segment) | Data Sci | Weekly | No disparate impact (>2x variance) |

| Monthly model retraining | ML Eng | Monthly | Retrained model passes staging tests |

Rollback Plan:

1. Trigger: Any critical failure (e.g., pipeline break, AUC <0.75).
2. Action:
 - DevOps switches to fallback model (<1 hr).
 - Comms lead alerts stakeholders (email template pre-approved).
3. Post-mortem: Root-cause analysis within 48 hrs (Data Sci + Eng).

Ownership Matrix

| Role | Key Responsibilities |

|-----|-----|

| Data Science | Model performance, fairness audits, RCA |

| Data Engineering | Pipeline reliability, feature store |

| DevOps | Deployment/rollback, monitoring infra |

| Growth | A/B tests, customer impact metrics |

| CX Team | Human review, complaint tracking |

Timeline Summary:

- Week 1-2: Validate and monitor.
- Week 3-4: Test with safeguards.
- Week 5+: Scale with rollback guardrails.

Why This Works:

- Clear escalation paths (no ambiguity on who acts).
- Incremental risk (shadow -> A/B -> full).
- Pre-defined failure modes (no debating when to roll back).

Engineering counterpart

3:30 AM - Responded in 0s

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

Agent response

3:31 AM - Responded in 16s

The #1 thing I'd want a second pair of eyes on:

Feature Pipeline Timing

Why?

- Highest risk area: Even small timing errors (e.g., a feature accidentally including 1 hour of future data) can silently corrupt the entire model.
- Hard to debug post-launch: Leakage often surfaces only after months, when weird patterns emerge (e.g., "why does `days\_since\_last\_login` perfectly predict churn?").

What to Double-Check:

1. Query/logic validation:

- Have a senior data engineer audit the SQL/Spark queries for features like `support\_tickets\_previous\_30d` to confirm:
 - They're strictly filtered on `event\_date < prediction\_date`.
 - No joins accidentally pull future data (e.g., a misaligned timezone in timestamps).

2. Backtest with frozen data:

- Simulate a prediction run for a past date (e.g., `2024-01-01`) using only data that existed at that time.
- Verify no feature values change when re-run later (if they do, there's leakage).

3. Log feature generation times:

- Add metadata like `feature\_computation\_timestamp` to predictions.
 - Spot-check that features for `prediction\_date=2024-06-01` weren't computed after midnight (e.g., `2024-06-01 00:05:00`).

Who Should Review:

- A data engineer with pipeline expertise (not the original developer, to avoid blind spots).
- Optional: A second data scientist to stress-test edge cases (e.g., new customers with no history).

Catchphrase for this review:

"If the feature computation ran at exactly midnight UTC on the prediction date, could it ever see the future?"

Why This Over Other Risks?

- Performance drops are visible and fixable later.
- Bias/fairness issues can be monitored post-launch.
- Pipeline timing errors are silent killers that invalidate everything.

(This is the hill I'd die on in a production review meeting.)

Engineering counterpart

3:31 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.