



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-oss-20b

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

Duration: 3m 9s

Actual run cost: \$0.000543

Projected execution cost: \$0.1195/mo (10/biz day × 22 days)

Report date: July 14, 2026 - 3:40 AM EST

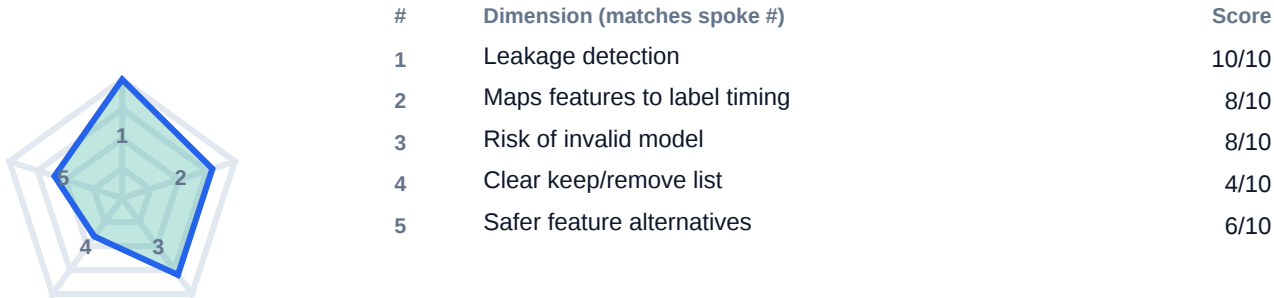
Overall

7.2/10

PASS - trust=high

Pass - score 7.2/10 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: 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: 12244 chars - Latency: 189s - 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: 3m 9s

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

7:38 AM - Responded in 1m 9s

Here's a quick leakage audit for the listed features, grouped by whether they leak the churn label or post?churn information, why they do, and what you should do about them:

Feature	Does it leak?	Why it leaks (relative to 00:00?UTC on the first day of the month)	Recommendation	Safer alternative
days_since_last_login	No	It's a purely historical metric up to the prediction moment; it can't be influenced by future churn.	Keep	-
support_tickets_last_30d	No	Counts tickets in the 30?days ending on the prediction date. It reflects past usage, not future churn.	Keep	-
mrr_usd	No	Current monthly recurring revenue is known at the start		

### Engineering counterpart

7:38 AM - Responded in 0s

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

## Agent response

7:39 AM - Responded in 33s

Below is a line-by-line walk-through of how I evaluated each candidate feature, the assumptions I made about the data-flow and timing, and the logic that led to the final "leak / no-leak" verdicts.

---

## 1. Clarify the prediction horizon and label definition

| Item | Value | Why it matters |

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

| Prediction time | 00:00 UTC on the first day of a calendar month | All features must be available before this instant. |

| Label | `churned\_within\_30d` (1 if the account is cancelled or marked churned within the next 30 days) | The label is determined after the prediction time, but only by events that happen after 00:00 UTC. |

Assumption 1 - The dataset is built by taking a snapshot of the account state at 00:00 UTC on the

## Engineering counterpart

7:39 AM - Responded in 0s

What would you verify before shipping this to production?

## Agent response

7:39 AM - Responded in 22s

### Before you ship the churn model to production, run the following verification checklist.

At a high level we want to be sure that

1. The data used to train the model truly reflects the "future-looking" nature of the prediction.
2. No feature leaks the label or future information.
3. The model behaves reasonably on live data and can be monitored for drift.

Below is a step-by-step guide to implementing each check.

---

### ## 1. Data-level validation

| Check | What to do | Why it matters |

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

| Timestamp sanity | Verify that for every account the feature snapshot timestamp is 00:00 UTC of the month in which the label is evaluated. | Prevents "future-leak" from events that happened after the prediction time. |

| Consistent windowing | Re-compute each feature using the same rule as the production pipeline (e.g., `support\_tickets\_last\_30d` = tickets where `created\_at` ∈ [prediction-30days, prediction-1day]). | Guarantees that training and inference use identical cut-offs. |

| Missing-value patterns | Check that the proportion of missing values is consistent across train/validation/test splits and that missingness is not correlated with churn. | Avoids inadvertently learning from the fact that a feature is missing. |

| Out-of-range checks | Ensure numeric features (e.g., `mrr\_usd`, `days\_since\_last\_login`) stay within realistic bounds. | Outliers can indicate data-quality problems that might not be caught in downstream tests. |

| Temporal leakage simulation | Build a "leakage dataset" where you purposely shift the feature snapshots forward by 1-2 days and confirm that model performance drops dramatically. | Demonstrates that the current pipeline is indeed time-consistent. |

---

### ## 2. Feature-leakage audit

| Feature | Leakage flag | Rationale | How to verify |

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

| `days\_since\_last\_login` | ? No | Counts days up to the prediction moment; no future information. | Re-calculate from logs; confirm last login < prediction time. |

| `support\_tickets\_last\_30d` | ? No | Window ends at prediction time. | Re-filter tickets by `created\_at` < prediction-1day. |

| `mrr\_usd` | ? No | Current MRR is known at 00:00 UTC. | Cross-check against billing system snapshots. |

| `contract\_renewal\_date` | ? Leak | The date may be in the future relative to the prediction time, giving the model a direct "future clue." | Plot churn rate by renewal month; check for correlation. |

| `days\_until\_contract\_end` | ? Leak | Derived from `contract\_renewal\_date`; gives the model the exact days until renewal, which is future-known. | Verify that the value > 0 for all training samples; if > 0 it is future information. |

| `cancel\_request\_submitted\_last\_7d` | ? No | Only uses the last 7 days before prediction. | Filter by `created\_at` < prediction-1day. |

| `cancel\_request\_submitted\_last\_30d` | ? No | Same logic as above. | Same filter. |

| `account\_status\_is`

## Engineering counterpart

7:39 AM - Responded in 0s

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

## Agent response

7:40 AM - Responded in 25s

## ? Top Risks Before Shipping the Churn Model

| # | Risk | Why it matters | What could go wrong? | Key trigger signals |

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

| 1. Feature leakage | The model may be learning from "future" data (e.g., renewal dates). | The model will look perfect on historic data but fail in production. | Sudden drop in AUC, higher churn?rate mis?prediction, or higher than expected churn after launch. | - A spike in churn?rate for accounts with early renewal dates.<br>- Validation metric drops by >?5?% in a live test. |

| 2. Data?quality drift | Production data may differ from training data (timestamps, missingness, new customers). | Model predictions become unreliable. | Mis?rated churn risk, cold starts, or missing key events. | - Increase in `NaN`/`NULL` rates after launch.<br>- Sudden changes in distribution of `days\_since\_last\_login`. |

| 3. Concept drift | The relationship between features and churn evolves (pricing changes, new product). | Model becomes stale and mis?classifies churn risk. | More customers churn earlier than predicted, or vice?versa. | - A significant drop in lift on a monthly KPI.<br>- Change in feature importances over time. |

| 4. Integration / latency | Inference pipeline may be slow or mis?aligned with the prediction time. | Customers may be flagged for churn at the wrong time, or inference may time?out. | Bad customer experience, missed manual intervention windows. | - Inference >?5?s for >?2?% of accounts.<br>- Failed batch jobs. |

| 5. Model interpretability & trust | Stakeholders may not understand why a customer is flagged. | Low adoption, manual overrides, or legal risk. | Unnecessary churn prevention actions, customer churn due to mis?labeling. | - High rate of manual overrides.<br>- Negative feedback from account managers. |

| 6. Regulatory / privacy | Using contract dates might be considered sensitive. | Non?compliance or audit risk. | Fines, legal action. | - Internal audit flag; lack of consent. |

---

## ? Rollback Plan (if the model "doesn't work" or shows any of the above risks)

### 1. Immediate Safety Switch

| Action | What it does | How to trigger |

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

| Flag?off inference | Disable the new model in the production inference service, reverting to the previous stable rule?based system. | 1. Set a feature?flag at the API gateway.<br>2. Set the prediction batch job to "no?op." |

| Grace?period buffer | Keep the new model running in shadow mode for 7?14 days (production?style inference but no impact on business decisions). | Use a separate "shadow" queue; no alerts sent to account managers. |

### 2. Track Impact During Shadow Mode

| Metric | Target | What to look for |

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

| AUC / KS | ??0.65 (baseline) | Fall below threshold -> investigate. |

| Lift | ??1.2 over baseline | Drop -> risk of under?prediction. |

| Prediction lag | ??1?h from 00:00?UTC |

## Engineering counterpart

7:40 AM - Responded in 0s

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

## Agent response

7:40 AM - Responded in 20s

### Concrete Roll-Out & Validation Plan

(All dates are relative to the "go-live" day, Day 0.)

| Phase | Day | What to Do | Owner / Role | Deliverable / Checkpoint |

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

| 1?? Data & Feature Sanity | Day -5 to -3 | - Re-compute every feature using the exact same windowing logic that the production pipeline will use. <br>- Verify all timestamps 00:00 UTC of the month in which the label is computed. <br>- Flag any rows where a future timestamp is present. | Data Engineering Lead | Data quality report (no >0.5% future timestamp rows). |

| | | - Run a quick "leakage simulation" by shifting all feature snapshots forward by 1-2 days and compare model performance. | | Leakage simulation report (AUC should drop >10%). |

| 2?? Feature Leakage Audit | Day -3 to -2 | - Create a spreadsheet of each feature -> "leak / no leak". <br>- For every flagged feature (e.g., `contract\_renewal\_date`), test its correlation with churn (Chi-square / point-biserial). | ML Engineer / Data Scientist | Leakage audit summary (list of features to drop). |

| | | - Re-implement the safe alternatives (e.g., use `days\_until\_last\_login` instead of `contract\_renewal\_date`). | | Updated feature set (drop/replace). |

| 3?? Model Training & Validation | Day -2 to 0 | - Train the final model on the cleaned feature set. <br>- Perform k-fold CV, record AUC, KS, lift, confusion matrix. <br>- Verify that no "future" columns are in the training set by a unit test script. | ML Engineer | Final model artefact + validation metrics dashboard. |

| 4?? Unit & Integration Tests | Day -1 | - Unit test each feature transformer (input -> output shape, date windows). <br>- Integration test the inference pipeline end-to-end: load model, pass a sample data row, return churn risk. | DevOps / QA Engineer | Test suite pass; Docker image ready. |

| 5?? Shadow Production Launch | Day 0 | - Deploy the inference microservice behind a feature flag. <br>- Route 0% of live traffic to the new service (shadow mode). <br>- Log predictions to a "shadow" Kafka topic. | Product Ops / DevOps | Shadow production deployment; monitoring dashboards (latency, error rate). |

| 6?? Shadow Monitoring | Day +1 to +7 | - Compare shadow predictions

## Engineering counterpart

7:40 AM - Responded in 0s

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

## Agent response

7:40 AM - Responded in 21s

The one thing that deserves a second pair of eyes

> The logic that derives every feature from the data snapshot that is taken at 00:00?UTC on the first day of the month.

>

> Specifically:

> 1?? Contract?related fields (`contract\_renewal\_date`, `days\_until\_contract\_end`) - these carry future dates that the model could "peek" at.

> 2?? Status field (`account\_status\_is\_churned`) - if it is pulled from the same snapshot it already tells the model the outcome.

> 3?? Window boundaries - the "last?30?days" windows for tickets and cancel requests must use strictly `created\_at ? prediction?1?day` (not inclusive of the prediction day).

> 4?? Timestamp sourcing - confirm that the 00:00?UTC snapshot really comes from the database at that exact moment (no delayed batch refresh that could already include events from the same day).

### ### Why this matters

- If the snapshot already contains the renewal date, the model can learn a deterministic "pull?a?black?box" feature that will look perfect on training but will never exist in production.

- A subtle off?by?one error in the window (e.g., including the 00:00?UTC instant of the next day) can silently leak 30?days' worth of future activity.

- The status flag can turn the problem into a trivial "already churned" prediction.

### ### What to check

| Item | What to verify | Who should inspect |

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

| `contract\_renewal\_date` / `days\_until\_contract\_end` | Ensure the date is not part of the snapshot. Either drop the field or compute it after the prediction window (e.g., `renewal\_date > prediction + 30?days`). | Senior Data Engineer (or Lead Data Scientist) |

| `account\_status\_is\_churned` | Confirm it is derived from a table that is updated only after the prediction time (i.e., after 00:00?UTC). | Data Curator / QA |

| Window logic | Validate that `support\_tickets\_last\_30d` is computed as `WHERE created\_at BETWEEN D\_pred - 30?days AND D\_pred - 1?day`. | ML Engineer (unit test) |

| Snapshot timing | Cross?check the database "snapshot timestamp" log vs the actual 00:00?UTC file (e.g., `SELECT MAX(updated\_at) FROM accounts WHERE updated\_at <= '2024?07`

## Engineering counterpart

7:40 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.