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-120b
Roles: Data Scientist (AI agent) <> Engineering counterpart (AI agent)
Duration: 2m 26s
Actual run cost: $0.000690
Projected execution cost: $0.1518/mo (10/biz day × 22 days)
Report date: July 14, 2026 - 3:40 AM EST

Session: sample-lb-rankings-seed-baddd7af-233a-4dc8-899e-c31f76e0f819

ds_feature_leakage_v1 - agent - openrouter - openai/gpt-oss-120b - ended

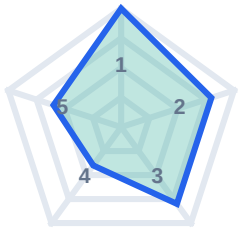
Overall

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: 15905 chars - Latency: 146s - 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: 2m 26s

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 24s

Quick?look summary

Feature	Leaks label / post?churn info?	Why it can leak (relative to prediction time = 00:00?UTC on the 1st of the month)	Recommended action	Safer alternative (if any)
`days_since_last_login`	No (if computed correctly)	Uses activity up to the snapshot moment. If the "last login" timestamp is after 00:00?UTC on the 1st, the derived value would be based on future behaviour -> leakage.	Keep but enforce that the login timestamp is ? the prediction timestamp.	N/A
`support_tickets_last_30d`	No (if window is anchored correctly)	Counts tickets that closed/were opened in the 30?days before the snapshot. If tickets filed after 00:00?UTC on the 1st are included, they belong to the "future" period and could already contain the churn decision (e.g., a ticket opened after a cancellation request).	Keep but truncate the window at the snapshot time.	N/A
`mrr_usd`	Potentially yes	If the MRR value is taken after the churn event (e.g., set to 0 when the account is already marked "churned"), the feature directly reveals the outcome.	Use the last known pre?churn MRR

Engineering counterpart
7:38 AM - Responded in 0s

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

Agent response
7:38 AM - Responded in 20s

Below is a step?by?step walk?through of how I examined each candidate column, the assumptions I needed to make, and the logical chain that let me decide whether the column can leak the churn label.

---

1. Clarify the prediction set?up

Element	What it means for the data timeline
Prediction moment	00:00?UTC on the 1st day of a calendar month (call this t?). The model will be trained on historic snapshots taken at exactly that moment and, when deployed, will be asked to predict inside the next 30?day window (t??->?t?+30?d).
Label	`churned_within_30d = 1` if the customer's account status changes to "churned" at any time in the interval (t?,?t?+30?d]. The label is therefore future information relative to the snapshot.
Training data	For each historical month we create a row that contains all information that was known at t? and the label that will be observed in the following 30?days. Anything that depends on events after t? is a potential source of leakage.

> Key assumption #1 - The "snapshot" we store for a given month is taken exactly at 00:00?UTC on the 1st, before any activity that occurs later that day.

---

2. General rule for each feature

1. Identify the temporal reference the feature uses (e.g., "last login", "last 30?days", "contract end date").
2. Determine the latest calendar date/timestamp that can influence the feature.
3. Compare that latest date with t?:
   If the latest possible timestamp ??t? -> the feature is historical and does not leak.
   If the latest possible timestamp >?t? -> the feature may incorporate future information, and we must check whether that future information already knows the churn decision.
4. Consider the business rule behind the metric (e.g., does a "cancel request" automatically trigger a churn status downstream?). If the metric is derived from a churn?triggering event, any inclusion of that event after t? is leakage.

---

3. Feature?by?feature reasoning

3.1 `days_since_last_login`

Step	Reasoning
What it measures	The number of days between the most recent login timestamp and the snapshot date.
Latest possible timestamp	The timestamp of the last login. If a login occurs at 12:00?UTC on the 1st, the "last login" would be after t?.
Potential leak?	Yes, if the calculation includes logins that happen after t? because a login that occurs after churn (e.g., a user logs in to cancel) would give a very low `days_since_last_login` and would be a proxy for the churn decision.
Assumption needed	We can guarantee that the last login we use is the most recent ??t?. In practice this means the ETL must filter out any login events with a timestamp >?t? before computing the feature.
Conclusion	Not inherently leaky, but only safe if the "last login" is truncated at the snapshot moment.

3

Engineering counterpart
7:38 AM - Responded in 0s
What would you verify before shipping this to production?

Agent response
7:38 AM - Responded in 32s
Below is a practical, end?to?end verification checklist you can run (or automate) before you push the churn?prediction model into a production environment.
The list is grouped by four logical pillars - data, model, monitoring, and process - and each bullet contains the why (the leakage?risk rationale) and how you can test it.

1?? DATA?LEVEL VERIFICATIONS

#	Check	Why it matters for leakage / label?timing	How to test / implement
1	Snapshot?time audit - confirm that every row in the training and inference tables is built from a view cut off at exactly 00:00?UTC on the 1st of the month.	Guarantees "future?looking" events (logins, tickets, cancellations) are never used as predictors.	- Add a hidden column `snapshot_timestamp` in the fact table (should be `2023?04?01 00:00:00`). - Write a Data?Quality (DQ) rule: `max(event_timestamp) ? snapshot_timestamp`. - Fail the ETL if any row violates it.
2	Feature?window truncation - for every "last?N?days" metric (`support_tickets_last_30d`, `nps_score_last_quarter`, etc.) verify that the window end is t? (the snapshot) and not t?+30?d.	Prevents including activity that happens after the prediction moment (e.g., a ticket opened on Jan?2 would be "future" for the Jan?1 snapshot).	- In the feature?generation SQL/ETL, compute `window_end = snapshot_timestamp`. - Unit?test: pick a random row, compare `max(ticket_created_at)` to `snapshot_timestamp`.
3	Churn?status leakage guard - ensure `account_status_is_churned` is always false at snapshot time for the training set; for inference rows it must be derived after the model score (i.e., set to "unknown").	If the column already reflects the churn flag at t?, the model would trivially learn the answer.	- Run a data sanity query: `SELECT COUNT() FROM training WHERE account_status_is_churned = 1 AND label = 0`. - Expect zero rows (or at most a handful of mis?aligned records that you will drop).
4	Derived?value sanity - for `mrr_usd`, confirm you are using the last non?zero MRR prior to churn (e.g., the value that existed on t?), not the post?cancellation MRR (which would be 0).	A zero MRR after churn automatically tells the model "this account left".	- Add a column `mrr_snapshot` = MRR as of `snapshot_timestamp`. - Compare to `mrr_at_churn_date`; they must differ only when `label=1`.
5	Cancellation?request timing - for `cancel_request_submitted_last_7d / 30d`, verify the underlying "request_timestamp" is ??t?.	A request submitted on Jan?3 would be in the "future" window for the Jan?1 snapshot and already encodes the churn decision.	- Filter out rows

Engineering counterpart
7:38 AM - Responded in 0s
What are the top risks, and what's the rollback plan if this doesn't work?

Agent response
7:39 AM - Responded in 34s

1?? Top Risks when you push the churn?prediction model to production

#	Risk	What could go wrong & why it matters (in the context of label?leakage / timing)	Early?warning signals (metrics / alerts)
1	Feature?leakage still present (a hidden "future" signal slips into a feature)	Even after the pre?deployment checks, a downstream ETL change (e.g., a new "login" event source, a revised ticketing schema) could start feeding timestamps after the snapshot into `days_since_last_login` or `support_tickets_last_30d`. The model would then appear to have "perfect" AUC on validation but would over?fit to the leakage and collapse on live data.	- Sudden jump in validation?to?production performance gap (e.g., AUC drops >?10?pts). - Spike in precision without a corresponding rise in recall (typical of a "cheat" feature). - Data?quality alerts: `max(event_timestamp) > snapshot_timestamp` for any feature table.
2	Label?definition drift (what we call "churn" shifts)	Today churn = "account_status becomes canceled within 30?d". If the product team later adds a "soft?pause" status or changes the billing cut?off, the historic label no longer matches the operational definition. The model will then be trained on the old definition and will mis?classify many accounts.	- Increase in "label?noise" metric: proportion of rows where `account_status_is_churned` at t?+30?d` ??business?rule?derived churn flag. - Manual audit of a random sample of churned accounts shows >?5?% mismatch.
3	Data?distribution shift (concept drift) - MRR, contract lengths, product usage patterns evolve	Even if the features are clean, the statistical relationships they capture may change (e.g., a new pricing tier makes `mrr_usd` less predictive). The model's ROC may degrade over weeks.	- Rolling?window performance dashboard: track AUC, precision@5?% and recall@5?% week?over?week. - Feature?distribution charts (e.g., KS?statistic) comparing last 7?days vs. training period.
4	Feedback loop / self?fulfilling prophecies	The model's "high churn risk" customers may be targeted by the retention team, causing them not to churn. This artificially inflates precision but reduces recall (the model never sees the "true" churn events it would have predicted). Over time the model can become under?confident and under?utilised.	- Decrease in observed churn rate among the top?risk bucket compared with historical baseline (while overall churn stays stable). - Retention?team reports "all high?risk accounts are being saved".
5	Infrastructure / latency failure - scoring service returns stale snapshots or fails to fetch the latest feature set	If the scoring pipeline falls back to an older snapshot (e.g., the previous month's feature dump), the model will be making predictions on out?of?date data, which can be especially	

Engineering counterpart
7:39 AM - Responded in 0s

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

Agent response
7:39 AM - Responded in 18s

Action?Plan to Ship the Churn Model??
Goal: Deploy a production?ready churn?prediction model that is free of label leakage and robust to drift.
The plan is broken into four phases (Data?Gate, Model?Gate, Go?Live, Post?Launch Guardrail) with clear owners, deliverables, and target dates (working weeks).

> Assumptions
> The team already has a data?warehouse (Snowflake/BigQuery) and a model?serving platform (Kubeflow?Serving, Sagemaker, or similar).
> All data engineers, data?science, and product?ops staff are co?located in the same Slack channel #churn?model?launch.

Phase	Week(s)	Owner(s)	Concrete Tasks & Acceptance Criteria
1?? Data?Gate - "Leak?Proof" Feature Store	W1?W2	Data Engineer Lead (DE?L) - Alex Data Quality Analyst (DQA) - Priya	1. Snapshot?time enforcement - Add a `snapshot_timestamp` column to the `customer_monthly_snapshot` table. 2. Write a DQ rule: `max(event_timestamp) ? snapshot_timestamp` for every source view (`logins`, `tickets`, `nps`, `invoices`). 3. Create unit?test pipelines (pytest + dbt tests) that fail if any future?dated event leaks into the snapshot. 4. Feature?window rewrite - Ensure all "last?N?days" metrics are built with `WHERE event_timestamp BETWEEN snapshot_timestamp - INTERVAL 'N days' AND snapshot_timestamp`. 5. Documentation - Add a feature?definition doc (Confluence) that explicitly lists the "cut?off = snapshot_timestamp". Acceptance: All dbt tests pass; a nightly monitoring job sends an alert if any row violates the rule (0 alerts for 2 consecutive runs).
2?? Model?Gate - Train & Validate on Clean Data	W2?W3	Data Scientist (DS) - Lena ML Engineer (MLE) - Ravi	1. Pull the clean snapshot (produced by Phase?1) into a training CSV. 2. Split chronologically: train on months???Nov?2023, validate on Dec?2023, test on Jan?2024. 3. Train baseline model (XGBoost/LightGBM). 4. Run feature?importance leakage scan: temporarily add a "future?leak" flag (e.g., `account_status_is_churned`) and verify its importance is???0. 5. Record performance metrics (AUC, PR?AUC, precision@5?%). 6. Model Card - Document data window, leakage checks, and assumptions. Acceptance: Validation AUC ??0.78, precision@5?% ??0.30, and the "future?leak" feature importance <?0.01.
3?? Go?Live - Deploy & Safeguard Scoring	W4	ML Engineer (MLE) - Ravi Platform Ops (POps) - Sofia Product Owner (PO)	

Engineering counterpart
7:39 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 18s

The single most critical thing to have a second set of eyes on is the "snapshot?cut?off logic" that guarantees every feature is built only from data that existed at or before 00:00?UTC on the 1st of the month.

Why?
All of the leakage risk in this project boils down to a tiny timing bug (e.g., a login or a cancel?request that occurs on Jan?2 sneaking into the "last?30?days" count for the Jan?1 snapshot).
If that gate is wrong, every downstream test (model performance, feature importance, monitoring) looks good on paper but will crumble as soon as the model scores live data.
The logic touches every upstream source (logins, tickets, invoices, NPS, contract dates) and is therefore easy to miss in a single person's code review.

What to ask the reviewer to verify

Step	What the reviewer should do	Expected outcome
1?? Confirm the cut?off timestamp variable	Open the ETL code (dbt model, Spark job, or SQL view) and locate the variable that represents the snapshot moment (e.g., `snapshot_ts`). Verify it is hard?coded to the start of the month (`DATE_TRUNC('month', current_timestamp)`) or passed in from the orchestration layer (Airflow, Dagster) and never overwritten later.	The variable is immutable for the whole run and matches `2023?04?01 00:00:00` for a April?2023 run.
2?? Check every source filter	For each source table (`logins`, `support_tickets`, `nps_responses`, `invoices`, etc.) ensure the WHERE clause ends with `event_timestamp <= {{ snapshot_ts }}` (or the equivalent).	No source contains `event_timestamp < {{ snapshot_ts }}` or a later date.
3?? Validate the "last?N?days" windows	Verify that a window such as "last?30?days" is expressed as `BETWEEN snapshot_ts - INTERVAL '30 days' AND snapshot_ts` - not `snapshot_ts + INTERVAL '30 days'`.	The SQL expression matches the pattern above.
4?? Run a sanity?query	Execute a quick ad?hoc query on the generated feature table, e.g.: `SELECT MAX(event_timestamp) FROM feature_table WHERE feature_name='days_since_last_login';` and compare the result to `snapshot_ts`.	The max timestamp must be ? `snapshot_ts` for all rows.
5?? Review the CI/CD test suite	Look at the dbt (or other) test files that enforce the rule (e.g., `assert max(event_timestamp) <= snapshot_ts`). Ensure the test is executed on every PR and that the CI pipeline fails on violation.	Test file exists, is part of the pipeline, and the latest CI run is green.
6?? Check the orchestration schedule	In Airflow/Dagster, verify that the DAG that materialises the snapshot runs just before the prediction job (e.g., at 00:05?UTC) and that the `snapshot_ts` argument is passed as a parameter, not derived later in the DAG.	DAG schedule = `0 0  ` (run at 00

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.