



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: inclusionai/ling-2.6-flash

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

Duration: 22s

Actual run cost: \$0.000153

Projected execution cost: \$0.0337/mo (10/biz day × 22 days)

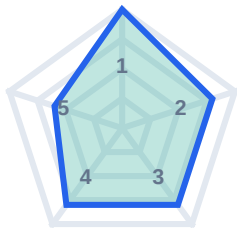
Report date: July 13, 2026 - 11:32 PM EST

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: 15128 chars - Latency: 22s - 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: 22s

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

Engineering counterpart

3:32 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

3:32 AM - Responded in 5s

Here is a focused leakage review for the churn task, with the prediction setup:

- Prediction horizon: first day of each month (00:00 UTC)
- Label: churned_within_30d (whether the customer churned within the next 30 days)

Features that leak the label or post-churn information

1) account_status_is_churned

- Why it leaks: This directly indicates whether the customer has already churned. At prediction time (start of month), this status either reflects past churn (if observed from historical snapshots) or, if current/reported in real time, it is the label itself.

2) cancel_request_submitted_last_7d

- Why it leaks: "Last 7 days" ending on the prediction day can include events after the prediction time. If a cancel request arrives after the start-of-month snapshot, using it leaks future information relative to prediction.

3) cancel_request_submitted_last_30d

- Why it leaks: Same issue as above: "Last 30 days" ending on the prediction day can include post-prediction events. Also, a cancellation request within 30 days before churn is often a direct consequence or marker of churn, creating label correlation.

4) days_until_contract_end

- Why it can leak: If contract end dates are determined or updated after the prediction date (e.g., renewals/cancellations negotiated after month start), this feature can implicitly depend on outcomes that occur after prediction.

5) mrr_usd

- Why it can leak: MRR reflects realized revenue; if a customer cancels or churns, realized MRR changes quickly. If MRR is updated with churn-related changes (failed payments, credits, downgrades) close to the prediction date, it leaks outcome information.

6) last_invoice_paid_amount

- Why it can leak: A missing or zero paid invoice near the prediction date can directly reflect non-payment or cancellation behavior that aligns with churn.

7) days_since_last_login

- Why it can leak: If computed in a rolling window that includes the prediction day or very recent activity, it can incorporate post-churn behavior (e.g., no login after churn).

8) support_tickets_last_30d

- Why it can leak: "Last 30 days" ending on the prediction day can include tickets logged after prediction; also, a spike in tickets often occurs post-churn or as churn happens.

9) nps_score_last_quarter

- Why it can leak: If NPS surveys are triggered by churn events or exits, or if the most recent NPS is collected very close to prediction and influenced by churn-related interactions, it leaks post-churn sentiment.

Note on account_status_is_churned: if this is a snapshot taken strictly before prediction (e.g., status as of last historical billing snapshot), leakage risk can be mitigated by ensuring no post-snapshot updates are included. In most production settings, it is safer to exclude it.

Recommendations: remove or recompute

- Remove or strictly time-align features that use "last N days" to end no later than the prediction timestamp.
- Avoid features that directly encode churn state or contractual states that update as a consequence of churn.
- Replace with time-safe proxies where feasible.

Safer alternatives and how to compute them

- Support tickets: use tickets_before_prediction (e.g., tickets in [prediction_N_days - 90, prediction_time)).

- Cancel requests: use `cancel_request_count_before_pred` (e.g., cancel requests before `prediction_time`).
- Days since last login: compute as `(prediction_time - last_login_before_prediction)`. Ensure `last_login`

Engineering counterpart
3:32 AM - Responded in 0s

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

Agent response

3:32 AM - Responded in 5s

Step-by-step reasoning and assumptions

1) Clarify the prediction setup

- I assumed the task is to predict, on the first day of each month, whether the customer will churn within the next 30 days.
- Prediction time = start of month, 00:00 UTC.
- Label = churned_within_30d (binary).
- "Churn" is typically defined as a termination/cancellation event or no active subscription after this 30-day window.

2) Define leakage criterion

- A feature leaks if its value at or before prediction time depends on information (labels, events, or measurements) that would not be available at prediction time in production.
- This includes:
 - Direct use of the label or its proxies.
 - Use of post-prediction events (even if unintentional).
 - Use of aggregates or snapshots that implicitly incorporate future churn-related changes.

3) Evaluate each candidate feature

- account_status_is_churned:
 - If this is a current status and can change because of churn (e.g., "churned", "at_risk", "active"), it is a direct proxy for the label.
 - Assumption: status can be updated as a consequence of churn or is the same state we are trying to predict. If it were a snapshot taken strictly before prediction (e.g., from a billing snapshot on day -1), leakage could be controlled, but that is fragile in practice.
- cancel_request_submitted_last_7d and cancel_request_submitted_last_30d:
 - "Last 7/30 days" windows ending on prediction day can include events after prediction if the window is computed relative to now rather than prediction_time.
 - Cancellation requests often increase as churn happens; they are strongly predictive but can leak label information.
- days_until_contract_end:
 - If contract end/renewal dates are updated after the prediction date (e.g., customer negotiates or cancels later), using this field can leak future outcomes.
- mrr_usd and last_invoice_paid_amount:
 - MRR and invoice payments can change quickly after churn (failed payments, credits, downgrades). If these fields are updated with churn-related events close to prediction, they leak outcome information.
 - Assuming "current" values may reflect churn-induced adjustments rather than pre-churn states.
- days_since_last_login:
 - If computed with a rolling window that extends to prediction_time, it can include post-churn inactivity.
 - Assumption: churn often leads to immediate login cessation; including very recent activity can couple the feature to the label.
- support_tickets_last_30d:
 - Similar to cancel requests; spikes often occur around or after churn. The "last 30d" window can include tickets logged after prediction.
- nps_score_last_quarter:
 - If NPS is surveyed or updated after prediction (or triggered by churn events), it can leak sentiment correlated with the label.

4) Key assumptions made

- Prediction time is exactly the start of the month and not a rolling "now". Features must be computable using data strictly before this timestamp.
- Churn events and their operational consequences (payment failures, status updates, contract changes) can occur after the prediction date and should not be included.
- Aggregates like "last 30d" are typically computed relative to "now" in practice, which can inadvertently include post-prediction data; I assumed they are defined relative to prediction_time unless proven otherwise.
- I assumed a typical B2B SaaS where cancellation, invoice, and login events can align with churn within short windows.

- I assumed contract and status fields can be updated by customer actions that follow churn-related decisions.

Engineering counterpart
3:32 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

3:32 AM - Responded in 4s

Before shipping, verify (1) data definitions and time boundaries, (2) feature computation pipeline, (3) leakage tests, (4) stability and operational risks.

Specific checks aligned to the features we discussed

- Exact definitions and availability time
 - Confirm for each feature: what it measures, its timestamp, and its update cadence.
 - Examples:
 - `account_status_is_churned`: when is it updated, and does it reflect churned state at prediction time or after?
 - `cancel_request_submitted_last_7d / last_30d`: are event timestamps recorded at initiation or at resolution? Is there a lag between event and ingestion?
 - `mrr_usd` and `last_invoice_paid_amount`: is MRR recomputed daily/weekly? Are failed payments or credits applied immediately?
 - `days_since_last_login`: is "last login" updated in near real time?
 - `support_tickets_last_30d / nps_score_last_quarter`: are these computed relative to `prediction_time` or "now"?
- Time boundaries and alignment to `prediction_time`
 - For every feature using a lookback window, enforce hard cutoff at `prediction_time` (first day of month, 00:00 UTC).
 - Implement window functions like `tickets_before_pred`, `cancel_requests_before_pred`, `login_max_before_pred` to guarantee no postprediction data can be included.
 - Validate that fields like `days_until_contract_end` do not depend on postprediction updates (prefer snapshot from before prediction or compute at scale with a fixed reference date).
- Label leakage checks
 - Confirm that none of the features directly or nearly duplicate the label:
 - `account_status_is_churned` should be excluded unless you can guarantee it's a prechurn snapshot.
 - `cancel_request_submitted_last_30d` should be excluded or recomputed as `cancel_request_count_before_pred`.
 - Run negative-control tests: compute feature distributions before churn and after random future dates; any strong separation indicates leakage.
- Data freshness and staleness
 - Confirm source systems provide data with latency low enough to meet "first day of month 00:00 UTC" SLA.
 - If any feature has >0 latency (e.g., daily batch load at 06:00), either shift prediction time or use prior-day snapshots.
- Feature stability and operational risks
 - Check for high missing rates or delays in any source system (e.g., NPS surveys arriving late, invoices not yet posted).
 - Monitor schema and cardinality changes (e.g., changes to contract status values or invoice structure).
 - Ensure deterministic, reproducible computation: same input data => same features.
- Model-level tests
 - Train a quick baseline with the engineered features and inspect performance uplift vs increasingly strict time windows.
 - Run leakage probes: train on future data and confirm performance drops to near-baseline.
 - Evaluate stability across time splits (e.g., month-by-month) to avoid temporal overfit.

Practical recommended actions before production

- Lock definitions and time cutoffs in documentation and SQL/code.
- Replace leaky features with safe versions:
 - `support_tickets_last_30d` -> `tickets_before_prediction(t0, window=30d)`
 - `cancel_request_submitted_last_7d/30d` -> `cancel_requests_before_prediction(t0)`
 - `account_status_is_churned` -> remove or use a prechurn snapshot
 - `mrr_usd/last_invoice_paid_amount` -> snapshot of mrr and invoicing status as of `t0`
 - `days_since_last_login` -> `(t0 - last_login_before_t0)`, with `last_login` capped before `t0`
- Add data tests asserting no postprediction timestamps enter features.
- Implement monitoring for data freshness

Engineering counterpart**3:32 AM - Responded in 0s**

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

Agent response**3:32 AM - Responded in 3s****Top risks**

- Label leakage in features: features like `account_status_is_churned` or `cancel_request_last_30d` can directly leak the label, causing overly optimistic validation performance that fails in production.
- Time-bound leakage: "last N days" features computed relative to "now" instead of `prediction_time` can include post-churn events, leaking future information at inference.
- Data freshness/latency: if source systems (invoices, logins, tickets, NPS) are not available by 00:00 UTC on the 1st, the model will miss features or fall back to stale defaults.
- Contract/MRR volatility: `days_until_contract_end` and `mrr_usd` can change after prediction due to renewals/cancellations, creating nonstationarity between training and inference.
- Negative controls failing: if engineered pre-churn signals (e.g., support volume, login activity) are too correlated with churn, the model may overfit and not generalize.
- Operational fragility: late-arriving data (e.g., delayed NPS or invoices) causing feature recomputation failures or backfills that break reproducibility.

Rollback plan

- Disable the new features; revert to the previous stable feature set that is known to work in production.
- If model versioning is in place, roll back to the last validated model version deployed before this feature set.
- If a full rollback is not feasible, disable or isolate the leaky features (e.g., replace rolling windows with before-prediction snapshots or remove them) and redeploy.
- Monitor key metrics (prediction latency, feature freshness, model performance/PSI) and alert on anomalies during the rollback.
- Document the root cause, update feature definitions and time-bound tests, and revalidate in a staging environment before re-release.

Engineering counterpart**3:32 AM - Responded in 0s**

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

Agent response

3:32 AM - Responded in 4s

Concrete next steps with owners and timing (immediate -> short-term)

Day 0-1 (immediate: stop/slow decisions)

- Freeze feature deployment to production. Owner: ML Engineering lead.
- Preserve the last known good model + feature set as rollback baseline. Owner: ML Engineering.
- Confirm prediction_time = 00:00 UTC on the 1st of each month and publish to stakeholders. Owner: PM/ML lead.

Day 1-3 (data & feature audit, high priority)

- Owner: Data Engineer + Data Scientist
 - Verify exact definitions and update cadence for each candidate feature (data dictionaries and source systems).
 - Confirm timestamps and ingestion latency for each source (login, invoice, ticket, NPS, contract dates, MRR).
 - Implement hard cutoff logic so all lookback windows use prediction_time, not "now".
 - Create SQL/data tests asserting no postprediction timestamps are included (unit tests + CI checks).

Day 1-7 (leakage validation)

- Owner: Data Scientist + Analytics
 - Run negative-control tests: compare feature distributions before real churn vs random future dates.
 - Train quick leakage probes to confirm no feature strongly predicts the label using only pre-prediction data.
 - Document and remove or recompute leaky features (cancel_request_last_7d/30d, account_status_is_churned, etc.) with safe variants.

Day 3-10 (rebuild & test)

- Owner: Data Scientist + ML Engineer
 - Recompute features with safe versions (before_prediction snapshots/windows).
 - Retrain baseline model with the safe feature set and evaluate on temporal splits.
 - Add monitoring for feature freshness, missing rates, and PSI drift.

Day 7-14 (staging & sign-off)

- Owner: PM + ML + Analytics
 - Stage validation: performance vs current model; ensure no leakage in backtests.
 - Data contract sign-off: owners confirm definitions, cadence, and rollback readiness.
 - Monitoring and alerting in production (freshness, missing values, prediction latency).

Ongoing (post-deploy)

- Owner: ML Ops/SRE
 - Deploy with feature flagging to enable quick disable.
 - Daily checks on feature availability and prediction job success; alert on delays > SLA.

Communication

- PM sends a brief status update at Day 1 and Day 7 to all stakeholders.
- Owners escalate immediately if leakage or freshness failures are detected.

If issues are found before deploy: pause rollout, fix definitions/tests, and revalidate. If issues are found after deploy: trigger rollback plan (disable new features/model, revert to prior stable version) and investigate root cause.

Engineering counterpart

3:32 AM - Responded in 0s

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

Agent response**3:32 AM - Responded in 1s**

The one thing I'd want a second pair of eyes on: the exact timestamp used for each feature's lookback window and confirmation that every "last N days" computation is strictly capped at prediction_time (first day of month, 00:00 UTC) with no access to postprediction data.

Engineering counterpart**3:32 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.