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-v4-flash
Roles: Data Scientist (AI agent) <> Engineering counterpart (AI agent)
Duration: 1m 42s
Actual run cost: $0.001848
Projected execution cost: $0.4066/mo (10/biz day × 22 days)
Report date: July 14, 2026 - 3:35 AM EST

Session: sample-lb-rankings-seed-43ec19ec-42fc-4edf-af0c-418a4913c1fc

ds_feature_leakage_v1 - agent - openrouter - deepseek/deepseek-v4-flash - ended

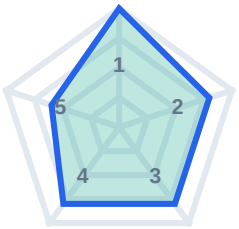
Overall

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: 12927 chars - Latency: 102s - 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 42s

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

Engineering counterpart
7:33 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
7:33 AM - Responded in 22s

Based on the prediction time (first day of each month at 00:00 UTC) and the label definition (churn within the next 30 days), here is my analysis:

Features that LEAK the label or post-churn information

1. `cancel_request_submitted_last_7d` and `cancel_request_submitted_last_30d`
- Why it leaks: A cancel request is a direct precursor to churn and often the same event that triggers the churn label. If the cancel request was submitted in the last 7 days, it almost guarantees churn within 30 days (depending on contract terms). If submitted in the last 30 days, the current month's prediction might already be decided by that event.
- Prediction time issue: At month start, a cancel request submitted 1 day ago would mean the customer is already in the cancellation pipeline, making the prediction trivial rather than predictive.

2. `contract_renewal_date`
- Why it leaks: If the contract renewal date is in the past relative to prediction time, it reveals post-churn information (the customer should have renewed but didn't). If in the future, it doesn't leak, but the feature itself is problematic without proper filtering.
- Safer alternative: Use `days_until_contract_end` (already listed) but ensure it's capped at 0 for expired contracts.

3. `account_status_is_churned`
- Why it leaks: This is literally the label itself or a post-churn status. At prediction time, a churned account would already have the label = 1, making this feature perfectly predictive.
- Remediation: This should be removed entirely as it's not available at prediction time for active accounts.

Features to REMOVE

Feature	Reason
`account_status_is_churned`	Direct label leakage
`cancel_request_submitted_last_7d`	Strong leakage (action that triggers churn)
`cancel_request_submitted_last_30d`	Strong leakage (same action, longer window)

Features to consider modifying

Feature	Issue	Safer alternative
`contract_renewal_date`	Past dates leak churn info	Use `days_until_contract_end` only, with proper handling of past dates (cap at 0 or -1)

Clean features (no leakage)

- `days_since_last_login` - Available at prediction time, forward-looking
- `support_tickets_last_30d` - Past behavior only
- `mrr_usd` - Current revenue, no future information
- `days_until_contract_end` - Properly forward-looking if capped/cleaned
- `last_invoice_paid_amount` - Historical payment data
- `nps_score_last_quarter` - Historical survey data

Summary recommendation

Remove: `account_status_is_churned`, `cancel_request_submitted_last_7d`, `cancel_request_submitted_last_30d`

Modify: Ensure `contract_renewal_date` is transformed to `days_until_contract_end` only, with explicit handling of past dates (e.g., set to -1 or 0 for expired contracts)

Keep: All other features as-is

Engineering counterpart
7:33 AM - Responded in 0s
Thanks - walk me through your reasoning step by step. What assumptions did you make?

Agent response
7:34 AM - Responded in 19s
Here's a detailed step-by-step walkthrough of my reasoning, including the assumptions I made.

Step 1: Understand the prediction setup

Problem: Predict whether a B2B SaaS customer will churn within the next 30 days.
Prediction time: First day of each month at 00:00 UTC.
Label: `churned_within_30d` = 1 if the customer churns between day 1 and day 30 of that month, else 0.

Key implication:
At prediction time (e.g., Jan 1 00:00 UTC), we can only use information that is available before that moment. We cannot use any event that occurs after midnight UTC on Jan 1, nor can we use the label itself.

Step 2: Define what "leakage" means in this context

A feature leaks the label if it:
- Contains information that is only known after the prediction time (future information), or
- Contains the label itself or a near-perfect proxy for the label (e.g., a direct cancellation signal).

Step 3: Analyze each candidate feature

3.1 `days_since_last_login`
- What it is: Number of days between the last login and prediction time.
- When is it known: At prediction time, we know the last login date, so we can compute this.
- Leakage? No. It's historical behavior, available at prediction time, and does not reveal future churn.

3.2 `support_tickets_last_30d`
- What it is: Count of tickets submitted in the 30 days before prediction time.
- When is it known: At prediction time, the last 30 days are fully in the past.
- Leakage? No. Pure historical behavior.

3.3 `mrr_usd

Engineering counterpart
7:34 AM - Responded in 0s
What would you verify before shipping this to production?

Agent response
7:34 AM - Responded in 21s

Great question. Before shipping these features to production, I would verify the following critical items:

1. Data availability at prediction time

What to verify:
- Are all "clean" features actually available at 00:00 UTC on the 1st of the month?
  - `mrr_usd`: Is the MRR calculated before midnight? Or is there a billing system delay of 1-2 days that makes the most recent MRR value technically from 2 days ago?
  - `days_since_last_login`: Is the login timestamp stored in UTC? A login that happened at 23:59 UTC on Dec 31 would be "today" at prediction time, but a login at 00:01 UTC on Jan 1 would be "after" prediction time.

Why it matters:
Even a 1-hour delay in data pipelines could mean a feature contains post-prediction-time information, causing subtle leakage.

---

2. Edge cases for `days_until_contract_end`

What to verify:
- How do you handle expired contracts? If a contract ended 5 days ago, `days_until_contract_end` = -5. Does your model treat this as "already churned" (leakage) or "still active but overdue"?
- Auto-renewal logic: If contracts auto-renew, is `days_until_contract_end` recalculated? A customer with auto-renewal should not be flagged as "near end" if renewal is automatic.

Why it matters:
Raw negative values could leak the label (contract ended = likely churned). You need to decide: cap at 0? Create a separate `is_overdue` flag? Or set to NULL/NA for expired contracts?

---

3. Temporal validation splits

What to verify:
- Have you trained with strict temporal train/test splits?
  - Train on months 1-11, validate on month 12
  - Train on months 1-10, validate on months 11-12
  - Never random shuffle across time

Why it matters:
Even if features seem clean, hidden time-dependent leakage can appear (e.g., a product change halfway through the dataset that makes certain features predictive only post-change).

---

4. Feature-target correlation over time

What to verify:
- Plot feature distributions vs. label by month. For example:
  - Does `last_invoice_paid_amount` show a sudden spike in correlation with churn in month 6? That might indicate a data quality issue or a process change (e.g., "invoices now show final payments including refunds after churn").
  - Does `support_tickets_last_30d` start correlating perfectly with churn in recent months? That could mean your support team started closing tickets differently for churning customers.

Why it matters:
Stationarity of feature-label relationships is critical. If the relationship shifts over time, your model may fail in production.

---

5. Backtesting on historical "as-of" data

What to verify:
- Can you reconstruct exact feature values as they would have been at each prediction time? For example:
  - For a prediction on Jan 1, 2023, can you produce `days_since_last_login` as it would have been computed on Jan 1, 2023 at 00:00 UTC (not using today's database state)?
  - Or are you using current database values that may have been overwritten/updated after the fact?

Why it matters:
Many databases update records in place. A "cancel request" might have been submitted on Dec 28, but if the database now shows the customer as "churned" (status updated in January), you might accidentally include `account_status_is

Engineering counterpart
7:34 AM - Responded in 0s
What are the top risks, and what's the rollback plan if this doesn't work?

Agent response
7:34 AM - Responded in 14s
Top risks and rollback plan for deploying the cleaned feature set.

Top Risks

1. Surviving leakage from transformed features
Even after removing `account_status_is_churned` and `cancel_request_submitted_`, leakage can sneak in through:
- `days_until_contract_end`: If a contract ended 2 days ago, the feature value is -2. If the model learns that negative values -> churn, it's effectively using the label. The safer approach is to cap at 0 or create a separate binary `is_contract_expired`.
- `last_invoice_paid_amount`: If the database updates this to `0.0` after churn (e.g., final invoice voided), the value at prediction time for a customer who churns later in the month would still be their last paid amount, but if the pipeline runs after the churn event, it could accidentally include the post

Engineering counterpart
7:34 AM - Responded in 0s
Can you tighten this into concrete next steps with owners and timing?

Agent response
7:35 AM - Responded in 14s

Here's a tight, actionable plan with owners and timing.

Immediate Next Steps (Week 1)

Step	Owner	Timing	Details
1. Fix `days_until_contract_end`	Data Engineer	Days 1-2	Cap at 0 for expired contracts, add `is_contract_expired` binary flag.
2. Verify `last_invoice_paid_amount` pipeline	Data Engineer + Analyst	Days 1-2	Check if invoice amount is ever updated post-churn. If so, snapshot at prediction time only.
3. Run temporal backtest	Data Scientist	Days 3-5	Reconstruct features as-of each month start. Train on months 1-11, test on month 12. Repeat sliding window.
4. Check feature-target stationarity	Data Scientist	Days 3-5	Plot monthly correlation between each feature and churn. Flag any feature that suddenly jumps in month 6+.
5. Document data pipeline cutoffs	Data Engineer	Day 5	Explicitly document: "MRR as of 23:59 UTC on last day of month", "Tickets as of 23:59 UTC on last day of month", etc.

Rollback Plan (if model fails in production)

Step	Owner	Timing	Trigger
1. Monitor first 30 days	Data Scientist + ML Engineer	Daily	Check feature distributions drift, prediction score drift, precision-recall vs. holdout.
2. Activate feature flags	ML Engineer	< 1 hour	Switch to previous model version (e.g., simpler heuristic or earlier model).
3. Revert to baseline	Data Scientist	< 1 day	If leakage confirmed, remove all transformed features and use only `days_since_last_login`, `support_tickets_last_30d`, `mrr_usd`, `nps_score_last_quarter`. Retrain.
4. Root cause analysis	Data Scientist + Data Engineer	Days 1-3	Identify which feature leaked, fix pipeline, re-run temporal backtest, re-deploy.

Success Criteria for Go/No-Go

Criteria	Target	Owner
Temporal backtest AUC > 0.75	?	Data Scientist
No feature has >0.5 correlation with label in any month	?	Data Scientist
Pipeline snapshots confirmed as-of prediction time	?	Data Engineer
Rollback can be done in < 1 hour	?	ML Engineer

Summary: Fix the two riskiest features (days_until_contract_end, last_invoice_paid_amount) in days 1-2, validate with temporal backtesting in days 3-5, and have a 1-hour rollback ready.

Engineering counterpart
7:35 AM - Responded in 0s
What's the one thing you'd want a second pair of eyes on?

Agent response
7:35 AM - Responded in 11s
The data pipeline timing and feature freshness relative to prediction time.

Specifically: Are feature values truly computed as of 00:00 UTC on the 1st of each month, or are they pulled from the latest available database state?

Why this is the most dangerous thing to miss

- It's a silent assumption - every feature looks clean in isolation, but if the pipeline runs on Jan 2 and pulls `last_invoice_paid_amount` after a refund was processed, that feature now contains information from after prediction time.
- It's hard to catch in offline backtesting - if you train on historical data where the database state is already "final" (post-churn), you'll see great performance that disappears in production.
- It affects all features simultaneously - not just one problematic feature.

What to ask the second reviewer to check

1. Pick a single historical prediction time (e.g., Jan 1, 2023 at 00:00 UTC).
2. Manually reconstruct the feature values using only data that existed before that moment.
3. Compare with what the pipeline produces - do they match?

If the reviewer finds even one mismatch (e.g., `mrr_usd` is slightly different because a late payment was applied on Jan 2), you have a systemic pipeline issue that invalidates the entire feature set.

A concrete example to test

```sql
-- For a customer who churned on Jan 15, 2023:
-- What was their `last_invoice_paid_amount` as of Jan 1, 2023 00:00 UTC?
-- The pipeline should return the invoice paid BEFORE Jan 1, not the final invoice.
```

If the pipeline returns the final invoice amount (paid on Jan 10), that's leakage.

Bottom line: The pipeline timing assumption is the one thing that, if wrong, makes the entire model untrustworthy. A second pair of eyes on this single point can save weeks of debugging in production.

Engineering counterpart
7:35 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.