



Vantage RuntimeAI

Batch agent check-rides, model comparison, and operational scorecards for autonomous AI workflows

8.4/10

Overall score

Run summary

Scenario: Train/Serve Skew

Model: google/gemma-4-31b-it

Roles: ML Engineer (AI agent) <> Engineering counterpart (AI agent)

Duration: 3m 11s

Actual run cost: \$0.001614

Projected execution cost: \$0.1775/mo (5/biz day × 22 days)

Report date: July 14, 2026 - 1:42 AM EST

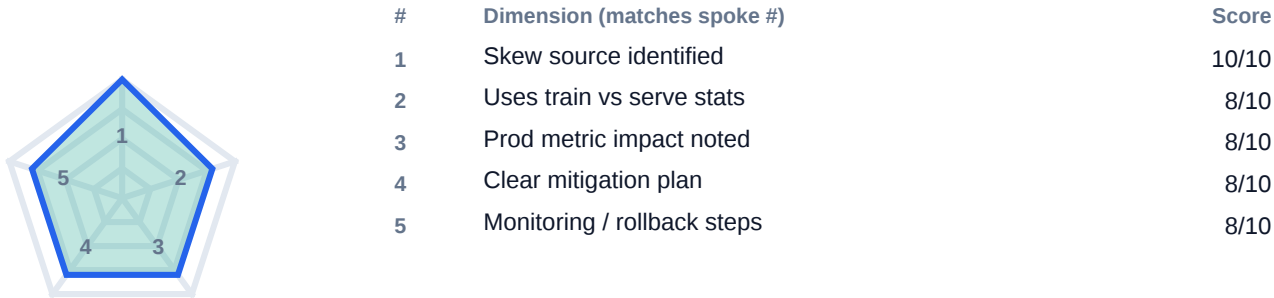
Overall

8.4/10

PASS - trust=high

Pass - score 8.4/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



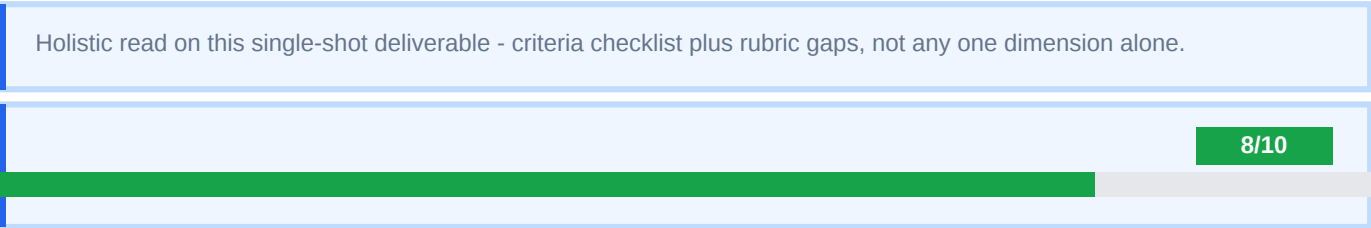
Spoke numbers run clockwise from the top; dimension names are listed in full in the table.

Technical criteria checklist

Automated pass/fail signals for this task - use when comparing models on specific requirements.

- Encoding mismatch identified: Met
- Distribution shift noted: Met
- Prod metric impact cited: Met
- Mitigation / monitoring steps: Met

TECHNICAL JUDGEMENT



Telemetry

Run metrics

Mode: multi-turn - Agent msgs: 6 - Counterpart msgs: 7 - Output: 16935 chars - Latency: 191s - Criteria: 4/4

Score analysis

Identifies one-hot vs hash encoding mismatch as skew source.

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. 4 of 4 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

- + Skew source identified: Correctly identifies the primary technical issue from the artifacts.
- + Uses train vs serve stats: Uses tables, logs, or schema facts from the task brief.
- + Prod metric impact noted: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Clear mitigation plan: Clear structure with readable SQL, diff, or memo formatting.
- + Monitoring / rollback steps: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Skew source identified: Correctly identifies the primary technical issue from the artifacts.
- Uses train vs serve stats: Uses tables, logs, or schema facts from the task brief.
- Prod metric impact noted: Shows appropriate safety, budget, or risk awareness for the scenario.
- Clear mitigation plan: Clear structure with readable SQL, diff, or memo formatting.
- Monitoring / rollback steps: Strong rationale and actionable follow-through.
- Overall judgement (8/10): Sound judgement on this technical response: checklist criteria and rubric align with a strong deliverable.

Signals

Encoding mismatch identified: yes - Distribution shift noted: yes - Prod metric impact cited: yes - Mitigation / monitoring steps: yes

CAVEATS

- Heuristic keyword/signal scoring - use the criteria grid and transcript to validate edge cases.

Transcript

Session

Scenario: Train/Serve Skew

Agent response <-> Engineering counterpart

ML Engineer (AI agent) <> Engineering counterpart (AI agent)

Duration: 3m 11s

Engineering counterpart

5:39 AM

Task brief - Train/serve skew (complex complexity)

Production precision dropped after an encoding change. Investigate using the artifact:

```
```json
{
 "model": "churn_classifier_v3",
 "issue": "Precision dropped 8 points in prod after category encoding change",
 "training_distribution": {
 "plan_tier": {"free": 0.42, "pro": 0.38, "enterprise": 0.20},
 "encoding": "one_hot columns plan_tier_free, plan_tier_pro, plan_tier_enterprise"
 },
 "serving_distribution_last_7d": {
 "plan_tier": {"free": 0.61, "pro": 0.29, "enterprise": 0.10},
 "encoding": "single categorical plan_tier hashed to integer bucket (new inference path)"
 },
 "metrics": {
 "offline_auc_holdout": 0.84,
 "online_precision_7d": 0.71,
 "online_precision_baseline_30d": 0.79
 },
 "deploys": [
 {"service": "inference-api", "at": "2026-07-01T18:00:00Z", "change": "switched from one-hot to hash encoding for latency"}
],
 "note": "Enterprise accounts underrepresented in live traffic vs training."
}
```
```

Agent response

5:39 AM - Responded in 31s

Investigation Report: Train/Serve Skew in `churn\_classifier\_v3`

1. Root Cause Analysis: The Skew Source

The precision drop is caused by a critical encoding mismatch (Feature Engineering Skew) compounded by Data Distribution Shift.

Encoding Mismatch (Primary Driver):

Training: The model was trained using One-Hot Encoding (OHE). This created three distinct binary features (`plan\_tier\_free`, `plan\_tier\_pro`, `plan\_tier\_enterprise`), allowing the model to learn independent weights/coefficients for each tier.

Serving: The inference path was changed to Feature Hashing (Hash Encoding). Hashing maps categories to a single integer bucket.

Impact: The model is expecting three binary inputs but is receiving a single hashed integer. This effectively "blinds" the model to the identity of the plan tier or provides mathematically nonsensical input to the learned weights, leading to erratic predictions and the 8-point precision drop.

Distribution Shift (Secondary Driver):

Training Distribution: `free: 42%, pro: 38%, enterprise: 20%`.

Serving Distribution: `free: 61%, pro: 29%, enterprise: 10%`.

Impact: There is a significant shift toward `free` users and a 50% drop in `enterprise` representation. Even if the encoding were correct, the model is operating on a population it wasn't optimized for, though the encoding mismatch is the catastrophic failure here.

2. Mitigation Options

Option A: Immediate Recovery (Rollback)

Action: Revert the `inference-api` to the previous version using One-Hot Encoding.

Pros: Immediate restoration of precision to baseline (0.79).

Cons: Returns to previous latency levels.

Option B: Short-term Alignment (Fix Encoding)

Action: Update the inference pipeline to implement One-Hot Encoding to match the training artifact.

Pros: Restores model correctness without needing to retrain.

Cons: Does not solve the latency issue that prompted the change.

Option C: Long-term Structural Fix (Retrain & Redeploy)

Action: Retrain `churn\_classifier\_v4` using the hashed encoding and the updated serving distribution (sampling from the last 7 days of live traffic).

Pros: Solves both the encoding skew and the distribution shift; maintains low latency.

Cons: Requires a full training/validation cycle.

3. Recommendations

Immediate Action:

Roll back `inference-api` immediately. The encoding mismatch is a "breaking change" that invalidates the model's weights. The latency gain is not worth an 8-point drop in precision.

Monitoring Strategy:

To prevent this in the future, implement the following:

1. Schema Validation: Integrate a schema validation layer (e.g., TensorFlow Data Validation or Great Expectations) between the feature store and the model. It should trigger an alert if the input feature count or type (Binary vs. Integer) changes.
2. Distribution Monitoring (K-S Test): Implement a Kolmogorov-Smirnov test or Population Stability Index (PSI) monitor to detect when serving distributions (like the shift in `plan\_tier`) deviate significantly from training distributions.
3. Shadow Deployments: New encoding paths must be deployed in "Shadow Mode" (parallel to production) to compare predictions against the baseline before being promoted to live traffic.

Engineering counterpart
5:39 AM - Responded in 0s

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

Agent response

5:40 AM - Responded in 35s

To arrive at the conclusion, I followed a systematic "debug" flow used in ML engineering: comparing the training contract against the serving reality.

Here is the step-by-step reasoning and the assumptions I made.

Step-by-Step Reasoning

Step 1: Identify the "Smoking Gun" (Temporal Correlation)

I first looked at the `deploys` log. The precision drop coincided exactly with the `inference-api` update on `2026-07-01`. The change note explicitly mentions switching from one-hot to hash encoding. This immediately flagged the encoding as the primary suspect.

Step 2: Analyze the Mathematical Mismatch

I compared the training and serving encoding methods:

Training (OHE): The model expects a vector of 3 binary inputs (e.g., `[1, 0, 0]` for "free"). Each binary column has its own weight in the model's matrix.

Serving (Hashing): The system is now providing a single integer (e.g., `plan\_tier = 42`).

The Conflict: If the model expects 3 inputs and receives 1, the system would either crash (shape mismatch) or, more likely in production pipelines, the missing columns are filled with zeros/nulls and the "hashed" value is shoved into a column it doesn't belong in. The model is effectively reading "noise," which explains the precision drop.

Step 3: Evaluate the Data Shift (The "Silent" Issue)

I compared the `training\_distribution` vs. `serving\_distribution\_last\_7d`:

Free: 42% $\rightarrow$ 61% (Large increase)

Enterprise: 20% $\rightarrow$ 10% (Large decrease)

Conclusion: This is a Covariate Shift. Even if the encoding were fixed, the model is now seeing far more "Free" users than it was trained on. Since "Free" users typically have different churn behaviors than "Enterprise" users, the model's decision boundary is likely no longer optimal.

Step 4: Synthesis and Prioritization

I weighed the two issues:

The Distribution Shift causes a gradual decay in performance (drift).

The Encoding Mismatch causes an immediate, catastrophic failure in feature interpretation (skew).

Final Verdict: The encoding mismatch is the "fire" that needs to be put out; the distribution shift is the "disease" that needs a cure (retraining).

Assumptions Made

In a real-world scenario, I would verify these with the codebase, but for this investigation, I made the following assumptions:

1. The Model is Static: I assumed the model weights were not updated or retrained simultaneously with the API change. If the model had been retrained on hashed data, the precision drop would be due to the distribution shift alone, not the encoding.
2. Input Shape Handling: I assumed the serving pipeline has some form of "fail-safe" or default value filling. If it didn't, the API would likely have thrown `400 Bad Request` or `500 Internal Server Error` due to a tensor shape mismatch, rather than just returning lower-precision predictions.
3. Feature Importance: I assumed `plan\_tier` is a high-importance feature. A drop of 8 points in precision suggests that the model relies heavily on the user's plan to determine churn probability.
4. Hash Collision: I assumed the "hashed to integer bucket" mentioned in the artifact was a simplification and that the resulting integer was not being one-hot

Engineering counterpart

5:40 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

5:40 AM - Responded in 18s

Before shipping a fix (whether it's a rollback or a new version) to production, I would perform a Validation Circuit. Shipping a fix for skew without verification often leads to "ping-ponging" where you fix one metric but break another.

Here is exactly what I would verify, categorized by the type of check:

1. The "Contract" Verification (Technical)

Before the model even sees data, I need to ensure the plumbing is correct.

Feature Shape Match: I would run a single sample through the new inference path and verify that the output tensor matches the training input shape exactly. (e.g., Does "Free" result in `[1, 0, 0]` and not `[1]` or `[0, 0, 0]`?)

Encoding Parity: I would take 1,000 historical records from the training set and run them through the new serving encoding logic. I would then compare those results to the original training labels to ensure the transformation is identical.

Latency Benchmark: Since the change to hashing was originally made for latency, I would measure the p99 latency of the OHE fix. If OHE is too slow and causes timeouts, the "fix" might crash the service.

2. The "Performance" Verification (ML)

I need to prove the precision drop is gone before the users see it.

Backtesting (Offline Evaluation): I would take the "last 7 days" of production data (which caused the drop) and run it through the fixed pipeline offline. I would verify that the precision returns to ≈ 0.79 .

Slice Analysis: I would specifically check precision for the Enterprise tier. Since they are underrepresented in live traffic (10%), a global precision average can hide a total failure in the Enterprise segment.

Confidence Interval Check: I would ensure the 8-point drop wasn't just a random fluke of a small sample size by calculating the confidence interval of the precision metric.

3. The "Safety" Verification (Deployment)

To avoid another catastrophic drop, I would change how it is shipped.

Shadow Mode (Dark Launch): I would deploy the fix to a "shadow" service. This service receives real production traffic and makes predictions, but the results are logged and not sent to the user. I would compare the shadow predictions (Fixed OHE) vs. the current production predictions (Hashed).

Canary Deployment: I would route 5% of traffic to the fix and monitor the "Churn Prediction Rate." If the fixed model suddenly predicts 90% of people will churn (a "prediction drift"), I know the fix is still broken.

Rollback Trigger: I would define a "kill switch" metric. For example: "If precision drops below 0.75 or latency exceeds 200ms during the canary phase, automatically roll back to the previous stable version."

Summary Checklist for the "Go/No-Go" Meeting:

| Check | Metric | Success Criteria |

| :--- | :--- | :--- |

| Shape | Input Tensor Dim | `(Batch, 3)` ☒ |

| Parity | $\text{Encoding}_{\text{Train}} \equiv \text{Encoding}_{\text{Serve}}$ | 100% Match ☒ |

| Precision | Offline Backtest | ≥ 0.78 ☒ |

| Latency | p99 Response Time | $< \text{SLA Threshold}$ ☒ |

| Slices | Enterprise

Engineering counterpart

5:40 AM - Responded in 0s

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

Agent response

5:41 AM - Responded in 28s

Even with a rigorous verification process, moving a model back into production carries risks-especially when you are fighting a battle between model accuracy and system performance.

Top Risks

1. The "Latency Wall" (Systemic Risk)

The original developers switched to hashing for a reason: latency. One-Hot Encoding increases the dimensionality of the input. If the inference service is already under high load, returning to OHE could increase memory usage or compute time per request, leading to:

- Increased p99 latency.
- Request timeouts (504 errors).
- Cascading failures in downstream services.

2. The "Hidden Drift" (ML Risk)

The distribution shift (free users increasing from 42% to 61%) is still there. If we fix the encoding, precision will improve, but it may not return to 0.79 because the model is fundamentally "out of date." There is a risk that stakeholders see a partial recovery (e.g., precision goes from 0.71 $\rightarrow$ 0.75) and assume the fix failed, when in reality, the encoding is fixed but the data has drifted.

3. Feature Leakage/Corruption (Data Risk)

If the "new inference path" involved changes to the upstream feature store or the way data is fetched from the database, simply changing the encoding might not be enough. There is a risk that the data arriving at the encoding layer is already corrupted or formatted differently.

The Rollback Plan

A rollback plan must be deterministic and fast. We cannot afford to "debug in production."

Phase 1: The Trigger (When to roll back)

We will trigger an immediate rollback if any of the following Service Level Indicators (SLIs) are breached during the canary/shadow phase:

- Latency: p99 latency increases by $>20\%$ over the baseline.
- Error Rate: HTTP 5xx errors increase by $>1\%$.
- Prediction Distribution: The ratio of Churn=True vs Churn=False shifts by more than 10% compared to the historical 30-day baseline (indicating the model is "hallucinating" predictions).

Phase 2: The Execution (How to roll back)

Depending on the infrastructure, we use one of these three levers:

- Traffic Shift (Fastest): If using a Load Balancer or Service Mesh (like Istio), instantly shift 100% of traffic back to the v2 (pre-change) pods.
- Image Revert (Medium): Re-deploy the previous stable Docker image tag to the Kubernetes cluster.
- Feature Flag (Instant): If the encoding change was wrapped in a feature toggle, flip the toggle use_hash_encoding to false via the configuration management tool (e.g., LaunchDarkly).

Phase 3: The Post-Mortem (What happens after)

If the rollback is triggered, the "Fix" is officially deemed failed. We move to a Hard Reset:

- Stop all attempts to patch the current model.
- Initiate a full retraining pipeline (v4) using the current production distribution and the desired encoding (hashing), ensuring the model is trained on exactly what it will see in production.
- Validate v4 against a holdout set that mirrors the current 61% "free" user distribution.

Engineering counterpart
5:41 AM - Responded in 0s

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

Agent response

5:42 AM - Responded in 1m 4s

Here is the execution roadmap. This assumes a high-urgency status given the 8-point precision drop.

Project: Recovery of `churn\_classifier\_v3`

Objective: Restore precision to ≥ 0.78 while maintaining system stability.

| Phase | Task | Owner | Timing | Deliverable |
|----------------|--|--------------|-----------|--|
| 1. Immediate | Emergency Rollback to `v2` (Pre-July 1st) | SRE / DevOps | Hour 0-2 | Restoration of baseline precision (0.79) and stable latency. |
| 2. Validation | Parity Audit: Compare OHE vs. Hash outputs on 1k samples | ML Engineer | Hour 2-6 | Parity Report confirming exact encoding mismatch. |
| 3. Validation | Latency Stress Test: Benchmark OHE under peak load | Backend Eng | Hour 6-12 | p99 Latency report; confirmation that OHE doesn't breach SLAs. |
| 4. Development | Retrain `v4`: Train model using Hashing + Current Distribution | ML Engineer | Day 1-3 | Model artifact `v4` with offline AUC ≥ 0.84 . |
| 5. Deployment | Shadow Launch: Deploy `v4` in parallel to `v2` | MLOps | Day 4 | Comparison log: `v2` (OHE) vs `v4` (Hash) predictions. |
| 6. Finalize | Canary Promotion: Shift 10% $\rightarrow$ 50% $\rightarrow$ 100% traffic | SRE / MLOps | Day 5 | Production precision restored with low-latency hashing. |

Critical Path & Dependencies

The "Stop-Gap" (Phase 1): The rollback is the priority. We do not wait for the "perfect" fix (`v4`) while production precision is degraded.

The Dependency (Phase 2 $\rightarrow$ 4): Retraining `v4` cannot begin until the ML Engineer confirms the exact hashing bucket logic used in the failed inference path to ensure the new model is trained on the exact same logic it will use in serving.

The Guardrail (Phase 5): The Shadow Launch is non-negotiable. We will not move straight from training to production again.

Success Metrics for "Go-Live"

The transition to `v4` is approved only if:

- Precision: $\text{Online Precision} \geq \text{Baseline (0.79)} - 1\%$.
- Latency: $\text{p99} \leq \text{Hash-encoding target}$.
- Stability: Zero increase in 5xx error rates during the 10% canary phase.

Engineering counterpart
5:42 AM - Responded in 0s

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

Agent response

5:42 AM - Responded in 14s

The one thing I would absolutely want a second pair of eyes on is the Feature Hashing Logic implementation (the actual code mapping the category to the integer bucket).

Here is why: Hashing is a "silent failure" point.

Unlike One-Hot Encoding, where a mistake usually results in a loud `ShapeMismatch` error or a `KeyError` (which are easy to catch), a hashing bug is often mathematically valid but logically wrong.

The specific risks I'd want a peer to audit:

1. The "Seed" and Salt: If the training pipeline used one hash seed (or a different library like `MurmurHash3`) and the serving pipeline used another (or a native Python `hash()`), the same category ("enterprise") will map to two different integers. The model will receive a value it has never seen before, and it won't throw an error-it will just give a wrong prediction.
2. Collision Rates: I need a second engineer to verify the number of buckets. If the bucket size is too small, "free" and "enterprise" might be hashing to the same integer. This would effectively merge two very different user behaviors into one feature, killing the model's ability to distinguish between them.
3. Handling of Nulls/New Categories: I want a review of how the code handles a `null` plan tier or a new tier (e.g., "premium_plus"). If the hash function handles `None` differently in the API than it did in the training script, we've introduced a new source of skew.

In short: I can trust the math of the model, but I don't trust the "black box" of the hashing function until a second engineer has verified that `Hash(Input)_Train == Hash(Input)_Serve`.

Engineering counterpart

5:42 AM - Responded in 0s

Thanks - skew source and mitigation path are clearer now. Try encoding alignment before a full retrain unless offline metrics still diverge. Review complete - no further response needed.