



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/gemini-2.5-flash

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

Duration: 32s

Actual run cost: \$0.007950

Projected execution cost: \$0.8745/mo (5/biz day × 22 days)

Report date: July 13, 2026 - 11:42 PM EST

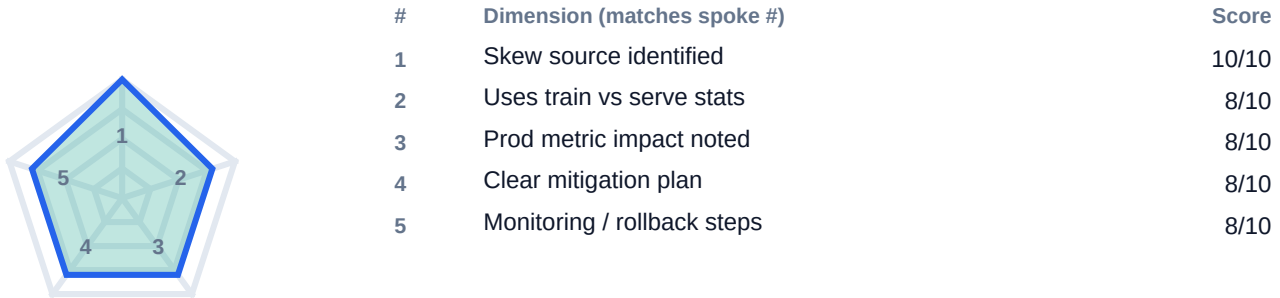
Overall

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

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: 20127 chars - Latency: 32s - 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: 32s

Engineering counterpart

3:42 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

3:42 AM - Responded in 5s

Here's an investigation into the train/serve skew, its potential causes, mitigation strategies, and monitoring/rollback recommendations:

Investigation Report: Train/Serve Skew - Churn Classifier v3

Model: churn_classifier_v3

Issue: Precision dropped 8 points in prod after category encoding change.

1. Analysis of the Skew Source

Based on the provided artifact, there are two primary sources of skew, both contributing to the observed precision drop:

a) Encoding Skew (Primary Cause)

Observation:

Training Encoding: `one\_hot` columns `plan_tier_free`, `plan_tier_pro`, `plan_tier_enterprise`

Serving Encoding: `single` categorical `plan_tier` hashed to integer bucket (new inference path)

Impact: This is a fundamental mismatch in how the `plan_tier` feature is represented between training and serving.

Training: The model was trained to interpret `plan_tier` as three distinct binary features. Each plan tier (free, pro, enterprise) corresponds to a unique input feature.

Serving: The new inference path hashes the categorical `plan_tier` string into a single integer bucket. This means the model is no longer receiving the distinct one-hot encoded features it was trained on. Instead, it's receiving a single, potentially non-linear, integer representation that it has never seen or learned to interpret during training.

Why it's critical: The model's weights associated with `plan_tier_free`, `plan_tier_pro`, and `plan_tier_enterprise` are effectively being bypassed or misinterpreted by the hashed integer input. This is akin to feeding entirely different features to the model than it expects.

b) Distribution Skew (Secondary/Aggravating Cause)

Observation:

Training Distribution `plan_tier`: `{"free": 0.42, "pro": 0.38, "enterprise": 0.20}`

Serving Distribution `plan_tier`: `{"free": 0.61, "pro": 0.29, "enterprise": 0.10}`

Note: "Enterprise accounts underrepresented in live traffic vs training."

Impact: Even if the encoding were consistent, the model would likely perform worse on "enterprise" users in production because it has seen fewer examples of them during training relative to their importance in production, and live traffic now has proportionally fewer enterprise users.

The model likely learned less robust patterns for "enterprise" users due to their lower representation in training.

The increased proportion of "free" users in serving, coupled with the decreased proportion of "enterprise" users, means the model is operating on a significantly different population than it was optimized for.

Synergy with Encoding Skew: The distribution skew makes the encoding skew even more damaging. If the model had learned strong, distinct patterns for `enterprise` users via one-hot encoding, those patterns are now completely lost due to the hashing. The change in distribution then means the model is disproportionately failing on categories it's now completely incapable of interpreting correctly.

2. Mitigation Options

Addressing the train/serve skew requires fixing both the encoding and ideally, understanding and adapting to the distribution shift.

a) Immediate Mitigation (Encoding Skew)

1. Rollback Inference Path (Highest Priority):

Action: Immediately revert the `inference-api` to use the original one-hot encoding pathway for `plan\_tier`.

Rationale: This directly addresses the primary

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

Agent response

3:42 AM - Responded in 5s

Okay, let's break down my reasoning step-by-step, including the assumptions I made.

Step-by-Step Reasoning

1. Deconstruct the Problem Statement

Core Issue: "Precision dropped 8 points in prod after category encoding change." This is the central piece of evidence. A clear causal link is implied.

Model: `churn\_classifier\_v3` - Contextual, but not directly involved in the skew analysis itself.

Key Data Points:

`training\_distribution`: Shows feature values and how they were encoded during training.

`serving\_distribution\_last\_7d`: Shows feature values and how they are currently encoded during serving.

`metrics`: Quantifies the problem (`online\_precision\_7d` vs `online\_precision\_baseline\_30d`). `offline\_auc\_holdout` provides a baseline of model quality before deployment issues.

`deploys`: Pinpoints the exact change and its timing, confirming the causal link. "switched from one-hot to hash encoding for latency."

`note`: An explicit observation about distribution differences.

2. Identify Discrepancies between Training and Serving

This is the core of train/serve skew detection. I'm looking for anything that's different between how the model saw data during training and how it's seeing data in production.

Encoding Mismatch:

Training: `one\_hot` columns `plan_tier_free`, `plan_tier_pro`, `plan_tier_enterprise`

Serving: `single categorical` `plan_tier` hashed to integer bucket (new inference path)

Reasoning: This is a major discrepancy. The model expects three distinct binary inputs (one-hot vectors like `[1,0,0]`, `[0,1,0]`, `[0,0,1]`). In serving, it's getting a single integer (e.g., `0`, `1`, `2`, or whatever the hash function outputs). The input feature space has fundamentally changed. The model's learned weights for `plan\_tier\_free`, `plan\_tier\_pro`, `plan\_tier\_enterprise` are now receiving completely different, or even garbage, input.

Distribution Mismatch:

Training `plan_tier`: `{"free": 0.42, "pro": 0.38, "enterprise": 0.20}`

Serving `plan_tier`: `{"free": 0.61, "pro": 0.29, "enterprise": 0.10}`

Reasoning: The proportions of each category have shifted. "Free" users are significantly more common in production, while "pro" and especially "enterprise" users are less common. The `note` explicitly confirms this focus on "enterprise" users. This means the model is operating on a different population mix than it was optimized for.

3. Prioritize Skew Sources

Encoding Skew vs. Distribution Skew: Which is more impactful?

Encoding Skew: This is a fundamental structural mismatch. It's like feeding a model expecting images an audio file. The model literally cannot interpret the input correctly. It's a "broken pipe" situation.

Distribution Skew: This is a "different data" situation. The model can interpret the input, but its performance might degrade because the data characteristics have changed, especially if the model performs worse on the underrepresented classes (like "enterprise").

Engineering counterpart
3:42 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

3:42 AM - Responded in 6s

Okay, this is a critical question for robust MLOps. Before shipping any changes (especially model updates or infrastructure changes affecting models) to production, I would verify the following, systematically and ideally through automated pipelines:

Foundational Checks (Pre-Deployment)

1. Code Review & Unit Tests:

What: New-feature development or bug fixes are peer-reviewed. All code paths (including encoding logic) have unit tests ensuring correctness on isolated components.

Why: Catches obvious bugs before integration. Ensures individual functions behave as expected.

2. Integration Tests:

What: Comprehensive tests that verify the interaction between different components of the inference pipeline (e.g., data fetching -> preprocessing/encoding -> model inference -> post-processing).

Why: Ensures the entire pipeline works together.

3. Data Schema Validation (Input/Output):

What: Define and enforce a strict schema for model inputs and outputs. Validate incoming inference requests against this schema.

Why: Prevents issues like missing features, incorrect data types, or unexpected categorical values, which can cause runtime errors or silent failures.

4. Model Loading & Availability:

What: Verify that the model artifact (e.g., ONNX, TensorFlow SavedModel) can be successfully loaded by the serving infrastructure. Check that the service starts up and exposes its endpoints correctly.

Why: Basic sanity check that the deployed artifact is valid and accessible.

Skew-Specific Verifications (Crucial for this Use Case)

5. Encoding Consistency Validation:

What:

Offline: Use a sample of production-like data (or recent production data) and run it through both the training-time encoding logic (as it was used to train the model) and the serving-time encoding logic. Assert that the transformed feature vectors are identical for all relevant features (e.g., `plan\_tier` in this case).

Online (Pre-deploy to Staging/Shadow): Instrument the serving code to log the actual encoded feature vector immediately before it's passed to the model. Compare this log with the expected encoded vector derived using the training-time logic.

Why: Directly verifies that the input the model receives in serving matches the input it was trained on. This is the most direct fix for the problem identified in the prompt.

6. Feature Distribution Consistency Validation (Targeted):

What:

Calculate feature distributions (histograms for numerical, counts for categorical) for the last `N` days of training data.

Calculate feature distributions for a representative sample of recent production data (or simulated production data).

Compare these distributions using statistical measures (e.g., Chi-squared test for categorical, Kolmogorov-Smirnov test for numerical, population stability index (PSI)).

Why: Identifies potential data drift or distribution skew before deployment. While the encoding was the primary culprit here, this check would have flagged the `plan\_tier` distribution shift even if encoding was consistent.

Performance & Quality Verifications

7. Offline Performance Validation on Holdout Data:

What: Evaluate the model's performance (precision, AUC, etc.) on a dedicated, untouched holdout dataset that reflects

the expected production data distribution.

Why: Ensures the model itself meets performance criteria before it even sees production traffic. This was ``offline_auc_holdout: 0.84`` in the prompt.

8. Prediction

Engineering counterpart
3:42 AM - Responded in 0s

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

Agent response

3:42 AM - Responded in 5s

This is an excellent follow-up question. Understanding risks and having a clear rollback plan is fundamental to responsible MLOps.

Top Risks

Given the historical issue and the proposed solutions, here are the top risks associated with deploying the corrected `churn\_classifier\_v3` and related infrastructure changes:

1. Incomplete Fix / Residual Encoding Bug:

Risk: We might think we've fixed the encoding discrepancy (e.g., reverting to one-hot), but there could be a subtle bug in the deployment process, configuration, or environment that reintroduces the wrong encoding, or a different mismatch. For example, if the one-hot encoder used in serving isn't exactly the same instance or configuration as was used in training, it could still cause issues.

Impact: Continued precision degradation, albeit potentially not as severe as the original 8-point drop, or a different kind of error.

2. Unforeseen Side Effects of Rollback/Change:

Risk: Reverting the `inference-api` to an older version or changing its encoding logic might introduce new, unexpected bugs in other parts of the service that we aren't directly testing (e.g., interaction with other features, logging, monitoring). The change to hash encoding might have been part of a larger `inference-api` update.

Impact: Service instability, other metrics degrading, or even total service outages.

3. Data Distribution Shift Persists/Worsens:

Risk: While the encoding fix addresses the immediate problem, the underlying shift in `plan\_tier` distribution (fewer enterprise users, more free users) remains. The model, even with correct encoding, might still perform suboptimally on this new distribution.

Impact: Performance might recover from the 8-point drop but might not reach the `0.79` baseline, or could even degrade again if the distribution shifts further.

4. Performance Regression (Latency/Throughput):

Risk: The original switch to hash encoding was "for latency." Reverting to one-hot encoding might reintroduce the latency or throughput issues that prompted the original change. This could impact user experience, API SLAs, or infrastructure costs.

Impact: Production system bottlenecks, increased response times, or higher cloud infrastructure bills.

5. Inadequate Monitoring & Alerting:

Risk: Even if the fix is correct, if our monitoring systems aren't robust enough or alerts aren't well-tuned, we might miss subtle regressions or the onset of new issues.

Impact: Prolonged suboptimal model performance, leading to continued business impact before detection.

Rollback Plan

A robust rollback plan is essential for minimizing the blast radius if an issue occurs.

Overall Goal: Revert to a known good state as quickly as possible, with minimal manual intervention.

Phases of Rollback:

Phase 0: Pre-Rollback Preparation (Before Deployment)

1. Define "Known Good State": Clearly identify the exact version of the `inference-api` service (container image, configuration) that was running before the problematic `2026-07-01T18:00:00Z` deploy, and ensure it's readily deployable. This is our primary rollback target.
2. Automated Rollback Scripts: Implement and test automated scripts/commands that can perform the rollback. This should be a single command or a short sequence.
3. Communication Protocol: Establish how the team will communicate if a rollback is triggered (e.g., Slack

Engineering counterpart
3:42 AM - Responded in 0s

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

Agent response

3:42 AM - Responded in 5s

Okay, let's transform this into a concrete action plan with assigned ownership and proposed timings. This assumes a standard agile workflow but can be adapted.

Project Title: Churn Classifier v3 Precision Recovery

Goal: Restore `churn\_classifier\_v3` online precision to baseline (0.79+) by resolving train/serve skew.

Current State: Precision at 0.71, down from 0.79 baseline due to encoding mismatch and distribution shift.

Key Personnel:

ML Engineer (MLE): Responsible for model-related code, data pipeline, monitoring definitions.

DevOps/Infra Engineer (DevOps): Responsible for deployment, service configuration, infrastructure.

Data Scientist (DS): Responsible for model evaluation, bias analysis, retraining strategy.

Product Manager (PM): Responsible for business impact assessment, prioritization.

Phase 1: Immediate Remediation & Verification (Target: 1-2 Days)

Objective: Quickly restore model precision by fixing the known encoding mismatch.

1. Action: Verify & Execute Rollback to One-Hot Encoding

Description: Confirm that reverting the `inference-api`'s encoding logic for `plan\_tier` back to one-hot encoding (as per training) is possible and directly addresses the identified encoding skew. This might involve deploying a previous version of the `inference-api` service or a targeted configuration change.

Owner: DevOps, MLE (joint)

Timing: Immediate (within 4 hours of approval)

Verification:

DevOps: Confirm successful deployment artifact (e.g., correct container image version).

MLE: Validate encoding in a staging environment using sample data (e.g., feed `plan\_tier='free'` and ensure `[1,0,0]` is the input to the model).

2. Action: Monitor Production Precision & Key Metrics

Description: Closely observe `online\_precision\_7d` and other relevant online metrics (e.g., AUC, recall, latency, error rates) post-rollback. Set up immediate alerts for any further degradation.

Owner: MLE, DevOps

Timing: Continuous for 24-48 hours post-rollback.

Verification: MLE confirms precision recovery towards baseline; DevOps confirms no new system errors or latency spikes.

3. Action: Document Rollback & Incident Report (if applicable)

Description: Record the details of the rollback, the identified cause, resolution steps, and immediate impact. If precision didn't immediately recover, initiate an incident report.

Owner: MLE

Timing: End of Day 1.

Phase 2: Reinforcement & Root Cause Analysis (Target: 3-5 Days)

Objective: Solidify the fix, prevent recurrence, and understand underlying causes.

1. Action: Implement Encoding Consistency Check in CI/CD

Description: Develop an automated test within the CI/CD pipeline that verifies the encoding transformation of `plan\_tier` (and ideally other categorical features) in the `inference-api` matches the training time encoding. This should run before deployment to production.

Owner: MLE, DevOps

Timing: Within 3 days.

Verification: New CI/CD job passes for the `inference-api` service.

2. Action: Enhance Data Monitoring for Feature Distributions

Description: Set up automated monitoring

Engineering counterpart

3:42 AM - Responded in 0s

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

Agent response

3:42 AM - Responded in 5s

The one thing I'd want a second pair of eyes on, and specifically from someone with deep domain knowledge or direct experience with the ``churn_classifier_v3`` model and its training pipeline, is:

"The exact implementation details of the original one-hot encoding used during training, and how we are ensuring the 'reverted' serving-side one-hot encoding exactly matches it."

Why this is critical and requires a second pair of eyes:

1. Subtlety of Categorical Encoders: One-hot encoding seems simple, but there are nuances:

Order of Categories: Did the original encoder sort categories alphabetically, or was there a specific order? If ``free``, ``pro``, ``enterprise`` resulted in ``[1,0,0]``, ``[0,1,0]``, ``[0,0,1]`` during training, but the new one-hot encoder processes them as ``enterprise``, ``free``, ``pro``, it would lead to a mismatch where ``enterprise`` might become ``[1,0,0]`` which the model interprets as ``free``.

Handling of Unknown Categories: What if a new ``plan_tier`` like "premium" appears? Did the original encoder error out, ignore it, or assign all zeros? The "reverted" encoder needs to behave identically.

Sparse vs. Dense Output: Was the original one-hot encoded output sparse or dense? While less common for such a small number of categories, it can matter for some model frameworks.

Feature Names/Column Order: The model expects specific input feature names (e.g., ``plan_tier_free``). Ensuring the revived one-hot encoding produces these exact column names in the correct order is crucial.

2. Risk of "Almost" Right: The biggest danger here isn't getting it completely wrong (which would likely cause an obvious error), but getting it almost right. If the encoding is slightly off, the model might still produce predictions, but performance will likely remain degraded, making the fix seem incomplete or leading to a frustrating debugging loop.

3. Source of Truth: Often, the one-hot encoder or its configuration is serialized and saved with the model during training. The best way to ensure consistency is to load that exact serialized encoder for serving, rather than recreating a new one. A second pair of eyes could help verify this approach, or if not possible, meticulously compare the code.

4. Avoiding Future Skew: This verification process itself can lead to better practices, such as serializing preprocessors alongside models (e.g., using ``pickle``, or saving a ``tf.feature_column.CategoricalColumn`` definition) to guarantee training-serving consistency.

By having a second person, particularly one familiar with the model's lineage, review this specific detail, we significantly increase the confidence that the encoding mismatch is truly and fully resolved.

Engineering counterpart

3:42 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.