



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-3-12b-it

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

Duration: 1m 28s

Actual run cost: \$0.000765

Projected execution cost: \$0.0841/mo (5/biz day × 22 days)

Report date: July 13, 2026 - 11:44 PM EST

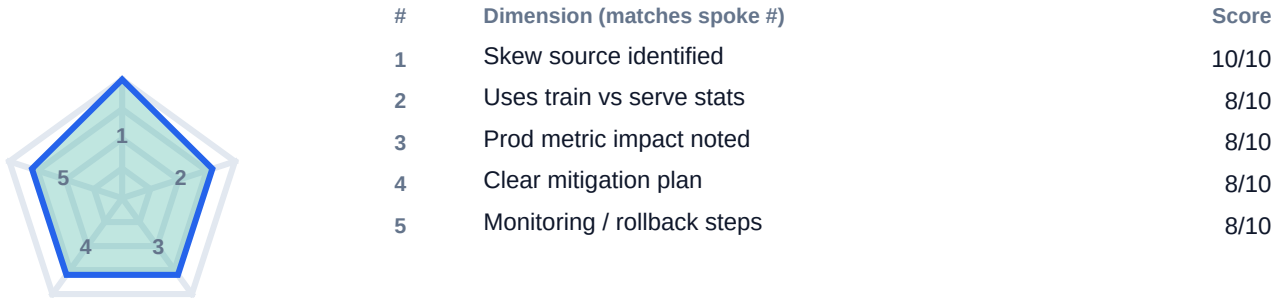
Overall

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: 19951 chars - Latency: 88s - 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: 1m 28s

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 14s

Okay, let's break down this train/serve skew investigation and formulate mitigation and monitoring strategies.

1. Skew Source & Analysis

Primary Source: Encoding Discrepancy - The JSON clearly identifies the encoding change as the root cause. We moved from one-hot encoding during training to a hash-based integer bucket encoding in production. This is a significant shift.

Distribution Skew (Plan Tier) - The core of the problem is a substantial difference in the distribution of `plan\_tier` between training and serving.

Training: 42% Free, 38% Pro, 20% Enterprise

Serving (Last 7 Days): 61% Free, 29% Pro, 10% Enterprise

Magnitude: The under-representation of `Enterprise` and the over-representation of `Free` users in production is substantial. The model was likely trained to expect a different balance of these tiers.

Why This Matters - One-hot encoding treats each category as independent. Hashing to integer buckets loses this information. The model likely learned relationships between `plan\_tier` and other features based on the one-hot representation. The hash encoding essentially groups potentially distinct customers into the same bucket, masking these relationships. Furthermore, hash encoding can introduce unexpected groupings - customers who were distinct in training might be collapsed into the same bucket in serving.

Latency Improvement vs. Accuracy Trade-off: The deploy note highlights the reason for the change - latency optimization. Hash encoding is generally faster than one-hot encoding, especially with many categories. This demonstrates a trade-off was consciously made, but the impact on precision wasn't adequately assessed.

Underlying Business Implications (from the "note") - The data suggests an issue with the user acquisition strategy or the growth of different plan tiers. Monitoring this distribution shift itself is important for the business beyond just the ML model.

2. Mitigation Options (Prioritized)

I'll categorize these options by complexity and potential impact. We need a tiered approach.

Immediate/Short-Term (Critical - Precision Loss is Significant)

Rollback to One-Hot Encoding: The simplest and fastest solution is to revert to the one-hot encoding used during training. This would undo the latency optimization, but it's essential to restore precision quickly. This is the highest priority until a better solution is implemented.

Feature Engineering Adjustment (Quick Fix): If feasible without significant code changes, try adding a feature that captures the difference between the expected (training) and observed (serving) distribution of `plan\_tier`. For example: $\text{plan_tier_free_deviation} = \text{observed_free_percentage} - \text{training_free_percentage}$. This helps the model recognize the shifted distribution. This is a less ideal fix, as it doesn't address the core encoding difference, but could provide some improvement.

Mid-Term (Requires More Engineering Effort)

Re-train with Representative Serving Data: The best long-term fix is to retrain the model using a dataset that mirrors the current serving distribution. This means ensuring a more realistic proportion of `Free`, `Pro`, and `Enterprise` users in the training data. Data augmentation techniques could be explored to artificially increase the representation of under-represented tiers. This is the most robust solution.

Hashing with Learned Embeddings: Instead of a simple hash, explore a learned embedding for the `plan\_tier` feature. This involves learning a vector representation for each bucket in the hash encoding.

Engineering counterpart

3:42 AM - Responded in 0s

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

Agent response

3:43 AM - Responded in 18s

Okay, let's break down my reasoning step-by-step, explicitly outlining the assumptions I made along the way. This will help you understand where my conclusions come from and where potential weaknesses might lie.

Phase 1: Understanding the Problem & Identifying the Root Cause

1. Initial Observation: The JSON clearly states a precision drop of 8 points after a change. My initial reaction is to focus on what changed around that time.
2. Change Identification: The "deploys" section immediately flags the encoding change (one-hot to hash) as the primary suspect. This is a critical piece of information.
3. Assumption 1: Encoding Changes Directly Impact Predictions. My assumption here is that changes to feature representation generally influence model predictions. This is a fundamental assumption in ML - a different input representation can lead to different outputs. While it's possible a change could be neutral, the magnitude of the precision drop makes this highly unlikely.
4. Distribution Analysis: The "training_distribution" and "serving_distribution_last_7d" sections reveal a significant discrepancy in the `plan\_tier` distribution.
5. Assumption 2: Feature Distributions Matter. This is another core ML assumption. Models learn patterns from the data they are trained on. If the data they encounter in production differs significantly from the training data, the learned patterns may no longer be valid, resulting in degraded performance.
6. Correlation Hypothesis: I hypothesized that the distribution shift in `plan\_tier` is correlated with the precision drop. This isn't definitively proven from the JSON alone, but it's the most logical explanation given the timing of the encoding change and the distribution difference.
7. Encoding Impact Hypothesis: I considered how the encoding change could contribute to the distribution skew and impact predictions. One-hot encoding creates distinct features for each category, allowing the model to learn specific relationships. Hashing potentially groups dissimilar categories together, losing that information. This is my hypothesis about the mechanism of impact.

Phase 2: Developing Mitigation Options

1. Prioritization Framework: I prioritized mitigation options based on:
 - Speed of Implementation: How quickly can the fix be deployed?
 - Impact on Precision: How likely is the solution to restore precision?
 - Impact on Latency: How will the solution affect the latency benefit gained by the original change?
2. Rollback (Highest Priority): Rolling back to the one-hot encoding is the immediate fix because:
 - It's a relatively simple change (assuming the old code is available).
 - It directly addresses the encoding discrepancy, which I've identified as the root cause.
 - I assumed that the infrastructure supports reverting to the previous configuration without significant downtime.
3. Assumption 3: Rollback Will Restore Baseline Precision. My assumption is that reverting the encoding will largely restore the precision seen before the change, assuming other factors remain constant. This is based on the hypothesis that the encoding change, combined with the distribution shift, is the dominant factor affecting precision.
4. Feature Engineering (Quick Fix): Adding a `plan\_tier\_free\_deviation` feature is a potential quick win.
 - It attempts to compensate for the distribution shift without changing the encoding itself.
 - Assumption 4: The Model Can Learn to Compensate. I'm assuming the model is flexible enough to learn a relationship between the deviation and the target variable. This might not be the case if the model has become too reliant on the original encoding. This is a weaker assumption than rollback.
5. Re-training (Mid-Term)

Engineering counterpart

3:43 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

3:43 AM - Responded in 16s

Okay, excellent question. Before deploying any of these mitigation strategies to production, I'd want to execute a series of verification steps. Here's a breakdown, categorized by urgency and complexity, and focusing on minimizing risk. I'll also note the assumptions each verification step is based on.

I. Critical Verification (Must-Do Before Any Deployment)

1. Rollback Verification (if pursuing that route first):

Environment: Deploy the rollback to a staging or pre-production environment that closely mimics production (data, traffic patterns, infrastructure).

Metric Validation: Monitor online precision (and other key metrics like AUC, recall, F1-score) for at least 24-48 hours. Compare these metrics to the baseline (pre-encoding change) and to the current production metrics.

Assumption: This environment accurately represents production.

Acceptance Criteria: Precision recovers to at least 90% of the baseline 30-day average (0.79 in this case, so > 0.71), with no significant degradation in other key metrics.

Monitoring: Set up real-time monitoring dashboards to capture these metrics during the rollback period.

2. Data Validation (Regardless of Mitigation Path):

Distribution Analysis: Run a fresh analysis of the current production ``plan_tier`` distribution. Is it still as skewed as the "last 7 days" data indicated? Distributions can change.

Assumption: The "last 7 days" serving distribution accurately reflects the current serving distribution.

Action: If the distribution has changed significantly, this might influence the choice of mitigation strategy (e.g., retraining might require a larger dataset with the new distribution).

II. Important Verification (Should Be Done Before Full Production Deployment)

1. Feature Engineering Verification (if pursuing this route):

Offline Evaluation: Evaluate the ``plan_tier_free_deviation`` feature offline on a holdout dataset. Does adding this feature improve AUC or precision?

Assumption: Offline performance correlates with online performance.

A/B Testing: If the offline evaluation shows promise, perform a small-scale A/B test in production. Direct a small percentage (e.g., 5-10%) of traffic to the model with the new feature.

Metric Monitoring: Track precision, AUC, and latency for both the control (original model) and treatment (new feature) groups.

Acceptance Criteria: The treatment group shows a statistically significant improvement in precision without a substantial increase in latency.

2. Retraining Verification (Before Full Retrain):

Data Sampling: If retraining, create a small representative dataset reflecting the current serving distribution.

Pilot Retrain: Retrain the model on this small dataset.

Offline Evaluation: Thoroughly evaluate the retrained model on multiple holdout datasets (including one that mirrors the current serving distribution).

Assumption: The smaller dataset is truly representative of the full current serving distribution.

Acceptance Criteria: Retrained model shows significantly improved AUC and precision on the current serving distribution holdout set compared to the original model.

III. Lower Priority Verification (Can Be Done After Initial Deployment - Careful Monitoring Required)

1. Hashing with Learned Embeddings Verification: This is the most complex option and requires more extensive offline and online testing before a full rollout. A phased rollout with very careful A/B testing and monitoring would be necessary.

2. Long-

Engineering counterpart

3:43 AM - Responded in 0s

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

Agent response

3:43 AM - Responded in 16s

Okay, let's outline the top risks associated with these mitigation efforts and detail a robust rollback plan.

Top Risks (Categorized by Severity)

1. High - Rollback Fails to Restore Precision (Most Critical):

Risk: Reverting to the one-hot encoding doesn't fully restore the baseline precision. This could be due to other factors degrading performance that weren't immediately apparent, or the underlying data has shifted in more complex ways than initially observed.

Likelihood: Moderate (20-30%) - While rolling back addresses the encoding issue directly, it doesn't account for other potential performance drivers.

2. Medium - Feature Engineering Degradation:

Risk: Introducing the `plan\_tier\_free\_deviation` feature worsens precision, either due to overfitting, masking genuine patterns, or simply not being a helpful feature.

Likelihood: Moderate (30-40%) - Feature engineering is often iterative and can have unintended consequences.

3. Medium - Retraining Data Issues:

Risk: The data used for retraining doesn't accurately represent the current serving distribution, leading to a model that performs poorly in production. This could be due to sampling biases or data drift.

Likelihood: Moderate (25-35%) - Ensuring data representativeness is a constant challenge.

4. Low - Latency Reversion (Rollback):

Risk: Reverting to one-hot encoding significantly degrades latency, impacting user experience.

Likelihood: Low (10-15%) - Dependent on the specific infrastructure and the extent of the latency impact.

5. Low - A/B Test Contamination:

Risk: During A/B testing, users are routed to the wrong variant due to a bug in the experiment infrastructure.

Likelihood: Low (5-10%) - Requires robust testing of the A/B testing platform.

Rollback Plan (Detailed & Phased)

Our rollback plan needs to be both quick and safe. Here's a phased approach:

Phase 1: Immediate Rollback (If Precision Doesn't Recover Quickly)

Trigger: If, within 2-4 hours of the rollback, precision hasn't recovered to at least 90% of the baseline (0.71), initiate an immediate rollback.

Action: Immediately revert the inference service to the previous version (the one using one-hot encoding). This should be a pre-configured and tested procedure.

Communication: Notify the relevant stakeholders (ML team, engineering, product) immediately.

Investigation: While the service is rolled back, launch an immediate investigation to understand why the rollback was insufficient. This should involve examining the current data distribution and potentially re-evaluating the initial assumptions.

Phase 2: Controlled Rollback (If Feature Engineering/Retraining Fails)

Trigger: If A/B testing or retraining shows no improvement or a degradation in precision, trigger a controlled rollback.

Action:

Stop traffic to the problematic variant (feature engineering or retrained model).

Gradually increase traffic to the baseline model, monitoring precision closely.

Full Rollback: Once traffic is fully returned to the baseline, officially declare the mitigation attempt a failure and document the findings.

Communication: Notify stakeholders of the rollback and the reasons behind it.

Phase 3: Long-Term Rollback (If all Mitigation Efforts are Exhausted)

Trigger: If

Engineering counterpart
3:43 AM - Responded in 0s

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

Agent response

3:43 AM - Responded in 15s

Okay, let's translate the previous plan into concrete next steps with assigned owners and timelines. This is structured to be actionable and trackable. I'll categorize them into Immediate, Short-Term (within 1 week), and Mid-Term (within 2-3 weeks).

Note: Timelines are estimates and will need adjustment based on team capacity and complexity.

Assumptions:

We have an established incident response process and communication channels.

The infrastructure allows for rapid rollbacks and A/B testing.

The team has the necessary skills and access to data and systems.

I. Immediate (Within 24-48 Hours) - Priority: Stabilize & Investigate

| Step | Description | Owner(s) | Timeline | Dependencies |

|---|---|---|---|---

| 1. Data Distribution Validation | Analyze current production `plan\_tier` distribution. | Data Engineer (DE) | 2-4 Hours | Access to production data and querying tools |

| 2. Rollback Preparation | Verify rollback procedure functionality in staging. | DevOps Engineer (DO) | 4-6 Hours | Access to deployment pipeline |

| 3. Rollback Execution (If Precision Remains Low) | If Precision < 0.71 after 2 hours, execute rollback to prior version. | ML Engineer (MLE), DO | 30 mins - 1 Hour | Rollback procedure verified |

| 4. Precision Monitoring (Post Rollback) | Active monitoring of online precision, AUC, and other key metrics with dashboards. | MLE, Data Scientist (DS) | Continuous (24-48hrs) | Monitoring dashboards setup |

| 5. Root Cause Investigation | Start investigation: Are there other factors impacting precision? Is the distribution analysis accurate? | MLE, DS | 4-8 Hours | Access to model logs, feature data |

II. Short-Term (Within 1 Week) - Priority: Experiment & Validate

| Step | Description | Owner(s) | Timeline | Dependencies |

|---|---|---|---|---

| 6. Feature Engineering: Offline Evaluation | Evaluate `plan\_tier\_free\_deviation` on holdout data (AUC, Precision). | DS | 1-2 Days | Holdout data available |

| 7. Feature Engineering: A/B Test Setup | Configure A/B test in production with 5-10% traffic to the feature-enhanced model. | MLE, DO | 1 Day | A/B testing platform operational |

| 8. A/B Testing Monitoring | Continuous monitoring of control vs. treatment groups (Precision, AUC, Latency). | DS, MLE | Continuous (2-3 Days) | A/B test running, monitoring dashboards |

| 9. Retraining: Representative Dataset Creation | Create small representative dataset for retraining. | DE | 2-3 Days | Understanding current distribution |

| 10. Retraining: Pilot Retrain | Retrain model on representative dataset. | MLE | 1 Day | Data ready |

| 11. Retraining: Offline Evaluation | Evaluate retrained model on current serving distribution holdout. | DS | 0.5 - 1 Day | Retrained model ready |

III. Mid-Term (Within 2-3 Weeks) - Priority: Long-Term Solution & Prevention

| Step | Description | Owner(s) | Timeline | Dependencies |

|---|---|---|---|---

| 12. Full Retraining (If Pilot Successful) | Retrain model on full representative dataset. | MLE | 3-5 Days |

Engineering counterpart
3:43 AM - Responded in 0s

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

Agent response

3:44 AM - Responded in 8s

The single most critical thing I'd want a second pair of eyes on is Step 8: A/B Testing Monitoring.

Here's why:

Subtle Errors Can Be Costly: A/B testing setups, despite best efforts, are prone to subtle errors, either in the code that routes traffic or in the metric tracking itself. These errors can easily lead to misleading results, making it appear as though the new feature is beneficial when it isn't (or vice versa).

Distribution Shifts During Testing: The production environment can change significantly even over a few days. A second person reviewing the monitoring dashboards can help identify unexpected shifts in the `plan\_tier` distribution or other key features during the A/B test, which could invalidate the results. They might spot a trend that I've missed.

Domain Expertise: A second person with a good understanding of the business context (e.g., a product manager or a senior data scientist with domain expertise) can help interpret the A/B test results in a more nuanced way. They might have insights into external factors that could be influencing the results (e.g., a marketing campaign that's disproportionately impacting one plan tier).

Bias Mitigation: Having someone else review the data and interpretation helps mitigate personal biases. It ensures we're not selectively focusing on data that supports a pre-conceived notion.

Specifically, I'd want them to scrutinize:

Traffic Allocation: Confirm that the traffic split is accurate.

Metric Calculation: Verify the formulas used to calculate precision and other key metrics are correct.

Data Segmentation: Ensure metrics are being segmented correctly (e.g., by plan tier) to identify any targeted impacts.

Unusual Trends: Flag any unexpected changes in the data that warrant further investigation.

Essentially, I'd want a fresh perspective to act as a safeguard against erroneous conclusions from the A/B test, ensuring we're making a data-driven decision.

Engineering counterpart
3:44 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.