



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: qwen/qwen3-235b-a22b-2507

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

Duration: 3m 16s

Actual run cost: \$0.002601

Projected execution cost: \$0.2861/mo (5/biz day × 22 days)

Report date: July 14, 2026 - 3:52 AM EST

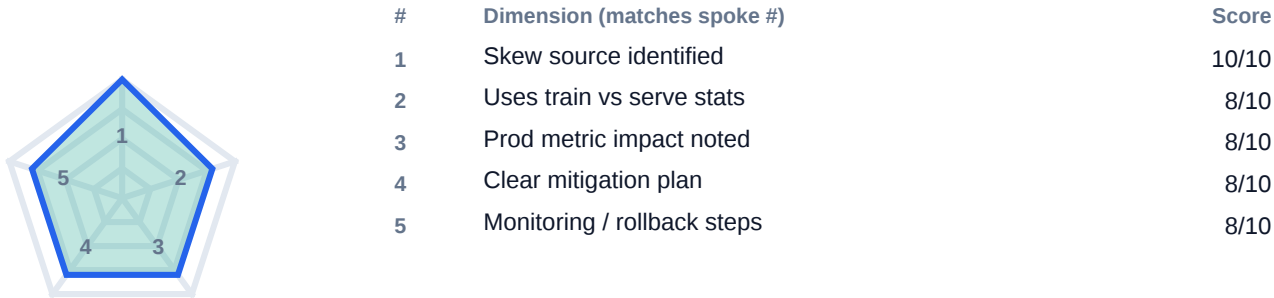
Overall

8.4/10

PASS - trust=high

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

CAPABILITY RUBRIC



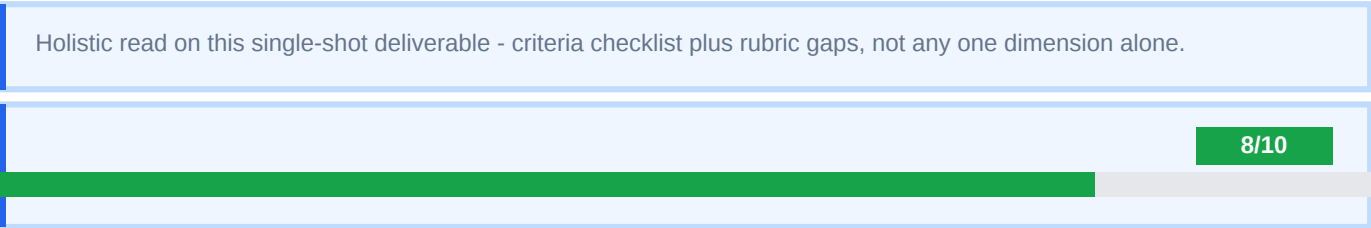
Spoke numbers run clockwise from the top; dimension names are listed in full in the table.

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: 18277 chars - Latency: 196s - 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 16s

Engineering counterpart

7:48 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

7:49 AM - Responded in 1m 5s

? ML Engineer Analysis: Train/Serve Skew Investigation

? 1. Skew Source: Confirmed - Feature Encoding Mismatch + Distribution Drift

The 8-point precision drop (from 0.79 -> 0.71) is caused by a train/serve skew with two compounding factors:

A. Encoding Skew (Primary Cause)

- Training: `one-hot` encoding with explicit columns:
`plan\_tier\_free`, `plan\_tier\_pro`, `plan\_tier\_enterprise`
-> Model learned distinct weights for each tier.
- Serving: `hash encoding` of `plan\_tier` into an integer bucket (e.g., using hash mod N).
-> Loses semantic meaning; introduces collisions and inconsistent mappings.

> ? Problem: Hashing is non-invertible and non-deterministic unless the hashing function and bucket size are perfectly synchronized between training and serving - and even then, it's fragile.

> - Hash collisions can map "enterprise" and "free" to the same bucket.

> - No guarantee that the encoded representation of "enterprise" is consistent with how it was learned during training.

This encoding mismatch means the model receives semantically different inputs at inference than during training -> high risk of incorrect predictions, especially for underrepresented classes.

B. Distribution Drift (Amplifying Factor)

- Training distribution:
`enterprise = 20%`
- Serving distribution (last 7d):
`enterprise = 10%` (50% drop)

> ?? Consequence: The model was trained to expect twice as many enterprise users as are now present in production.

> But crucially, due to the encoding degradation, the signal for identifying enterprise users is now noisy or lost -> precision drops sharply when trying to predict churn (likely false positives increase).

This is consistent with the note: "Enterprise accounts underrepresented in live traffic vs training."

? Why Precision Dropped (Not Recall or AUC)?

- AUC unchanged offline (0.84) -> Model still ranks well on held-out data using correct encoding.
- Precision dropped online -> Indicates increased false positives in production.
- With hash encoding, rare classes like "enterprise" may be misencoded -> model misclassifies them as high-risk (churn), but they don't churn -> false positives rise -> precision drops.

? Mitigation Options

| Option | Description | Pros | Cons |

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

- | 1. Rollback encoding (Immediate) | Revert inference API to use one-hot encoding for `plan\_tier` | Fast, restores performance, eliminates skew | Slight latency increase; doesn't fix root cause |
- | 2. Align encoding in both paths | Retrain model using same hash encoding as serving | Eliminates train/serve skew | Requires retraining, validation, delay; hashing still risky |
- | 3. Use learned embeddings or ordinal encoding | Replace one-hot/hash with embedding layer or ordered encoding (if applicable) | Better scalability, avoids one-hot bloat | Requires architectural changes; may need more data |
- | 4. Hash with consistent namespace & salt | Use deterministic hashing (e.g., `farmhash('plan\_tier\_enterprise') % N`) with fixed bucket size | Reduces collisions if done carefully | Still loses semantics; error-prone |
- | 5. Shadow mode validation | Run both encodings in parallel, compare outputs | Detects skew without user impact | Adds infra complexity; not a fix |

> ? Recommended Immediate Action:

Engineering counterpart
7:49 AM - Responded in 0s

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

Agent response

7:49 AM - Responded in 9s

Absolutely - let's walk through this step by step, with full transparency on observations, inferences, and assumptions. We're acting as an ML engineer diagnosing a production issue with limited data - so we must be rigorous.

? Goal:

Determine why precision dropped 8 points in production after a change, given the artifact.

We are told:

> "Precision dropped 8 points in prod after category encoding change"

We must:

1. Identify the source of train/serve skew
2. Understand the role of data distribution
3. Uncover any hidden assumptions we're making
4. Recommend actionable fixes

? Step 1: Compare Training vs. Serving Environments

Let's extract key facts from the JSON:

? Training Configuration

- Feature: `plan\_tier`
- Encoding: One-hot encoding
 - Three binary columns:
`plan\_tier\_free`, `plan\_tier\_pro`, `plan\_tier\_enterprise`
- Distribution of `plan\_tier`:
`free: 42%`, `pro: 38%`, `enterprise: 20%`

-> Model was trained seeing explicit, sparse vectors for each user's plan tier.

Example:

```
```python
Free user -> [1, 0, 0]
Pro user -> [0, 1, 0]
Enterprise user -> [0, 0, 1]
```
```

The model learns independent weights for each of these features.

?? Serving Configuration (Now in Prod)

- Feature: `plan\_tier`
- Encoding: Hashed to integer bucket (e.g., `hash(plan\_tier) % N`)
- Distribution:
`free: 61%`, `pro: 29%`, `enterprise: 10%`

-> Now inputs are a single integer derived from hashing the string category.

Example:

```

python
hash("free") % 10 -> 3
hash("enterprise") % 10 -> 3 <- collision!

```

So both "free" and "enterprise" might map to the same value -> model sees identical input for different categories.

? Step 2: Was There a Train/Serve Skew?

Yes - clear mismatch.

| Aspect | Training | Serving |
|------------------|-------------------|------------------------------------|
| Encoding | One-hot (3 cols) | Hashed integer (1 value) |
| Input dim | 3 | 1 (or embedded scalar) |
| Semantic clarity | High (orthogonal) | Low (ambiguous, collisions likely) |
| Distribution | enterprise = 20% | enterprise = 10% |

?? This is a classic train/serve skew: the model is trained on one representation, but served using a different one.

> Even if the model could work with hashed features, it wasn't trained that way.

? Step 3: Why Did Precision Drop Specifically?

Metrics:

- Offline AUC (holdout): 0.84 -> still strong
- Online precision (baseline): 0.79
- Online precision (post-deploy): 0.71 -> 8.0% absolute drop

But note: AUC is not the same as precision.

- AUC measures ranking quality (how well model separates classes).
- Precision measures: of all users predicted to churn, how many actually churned?

So a drop in precision suggests:

> The model is now predicting more false positives - more non-churners are being flagged as churners.

Why?

Let's hypothesize:

Hypothesis:
Enterprise users are less likely

Engineering counterpart
7:49 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

7:50 AM - Responded in 29s

Excellent question.

Before shipping any model or feature pipeline change - especially one involving encoding transformations, distribution shifts, or infrastructure changes (e.g., latency optimizations) - we must verify a set of critical properties to prevent train/serve skew, performance degradation, or silent failures.

Below is a structured pre-production verification checklist, tailored to this incident, with specific tests, tools, and validation points.

? Pre-Production Verification: 5 Key Areas

1. Feature Encoding Consistency (Train/Serve Alignment)

> ? Verify: The exact same transformation is applied in training and serving.

What to Check:

- Are one-hot categories encoded identically in batch and online?
- If using hashing:
 - Is the hash function deterministic? (e.g., `hashlib.md5`, `farmhash`)
 - Is the bucket size fixed?
 - Are collisions logged or monitored?
 - Was the model retrained on hashed features if that's what's used in serving?

How to Verify:

| Method | Tool/Approach |
|------------------------|---|
| Shadow Encoding Test | Run live traffic through both old and new encoders; compare outputs |
| Unit Tests | Test encoder on known inputs:
`"enterprise"` -> should always produce same hash bucket |
| Model Input Inspection | Log raw model inputs (tensors) in staging; compare train vs. serve |
| Data Contract | Define schema + transformation logic in code (not config drift) |

> ? In this case: No evidence the model was retrained on hashed inputs -> invalidates assumption that hashing can be swapped in safely.

2. Feature Distribution Drift Detection

> ? Verify: The data seen in production is within expected bounds of training data.

What to Check:

- Are category frequencies in serving within $\pm 15\text{-}20\%$ of training?
- Is any class (e.g., `enterprise`) becoming rare in prod?
- Is there concept drift? (e.g., enterprise users now behave differently)

How to Verify:

| Method | Tool/Approach |
|--------|---------------|
| | |

| Drift Monitoring | Use `Evidently`, `NannyML`, or custom stats (PSI, KL divergence) |
| Alerting on Class Imbalance | Anomaly detection on `plan\_tier` distribution |
| Slice Validation | Evaluate model performance per `plan\_tier` group in staging |

> ? Finding from data: `enterprise` dropped from 20% -> 10% -> outside safe range, especially for model relying on strong signal from that group.

3. End-to-End Model Behavior (Staging with Real Traffic)

> ? Verify: The full pipeline (featurization -> model -> prediction) behaves as expected on real user data.

What to Check:

- Do predictions match between staging and prod mock?
- Are edge cases (e.g., new plan tiers, missing values) handled?

How to Verify:

| Method | Tool/Approach |
|-----|-----|
| Shadow Mode | Deploy new model/encoder alongside current - log outputs but don't serve |
| Canary Comparison | Compare predictions from old and new path on same input |
| Prediction Diff Testing | Flag inputs where predictions diverge > threshold |
| Data Logging | Capture features, encoded tensors, predictions in staging |

> ? Example:

> Input: `{ "plan\_tier": "enterprise" }`
> - Old: one-hot -> [0,0,1] -> churn prob = 0.3
> - New: hash -> 7

Engineering counterpart
7:50 AM - Responded in 0s

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

Agent response

7:50 AM - Responded in 20s

Excellent - moving from diagnosis to risk management and operational resilience is critical in ML systems.

Based on the current state (encoding change deployed, precision dropped), here's a comprehensive breakdown of:

1. Top Risks
2. Rollback Plan
3. Rollback Triggers (When to Activate)
4. Post-Rollback Actions

?? 1. Top Risks (Immediate & Latent)

| Risk | Severity | Impact | Likelihood | Notes |
|--|----------|--|------------|------------------------------------|
| Persistent train/serve skew | ? High | Wrong predictions -> loss of trust, business impact | High | Current root cause |
| False positives in churn predictions | ? High | Over-targeting loyal users (e.g., enterprise) -> poor CX, wasted spend | High | Precision drop confirms this |
| Silent model degradation | ? High | No alerts on encoding drift -> issue persists | Medium | Common in ML ops |
| Hash collisions misrepresenting segments | ? High | "Free" and "Enterprise" map to same bucket -> model can't distinguish | High | Especially if bucket size is small |
| Retraining on mismatched pipeline | ? Medium | Future retraining still uses one-hot, but serving uses hash -> skew persists | Medium | Without alignment, cycle repeats |
| Latency vs. accuracy trade-off backfires | ? Medium | Hashing added for latency, but if model becomes unusable, trade-off invalid | Medium | Optimization failed |
| Cascading impact to downstream systems | ? Medium | Churn predictions feed retention emails, support routing, etc. | Medium | Business logic may amplify errors |

> ? Key Insight: This isn't just a modeling issue - it's a systemic risk involving data, infrastructure, monitoring, and business logic.

?? 2. Rollback Plan

> Goal: Restore precision and eliminate train/serve skew quickly and safely.

? Step-by-Step Rollback

| Step | Action | Owner | Tooling |
|------|---|-------------------|---------------------------------------|
| 1 | Trigger rollback if precision < 0.75 for 24h or alert fires | ML Engineer | Monitoring system |
| 2 | Re-enable old inference path with one-hot encoding | ML Ops | Kubernetes config / feature flag |
| 3 | Verify encoding in serving logs | SRE / ML Engineer | Log sampling (`grep "plan_tier_free") |
| 4 | Confirm prediction alignment with pre-change baseline | Data Scientist | Shadow comparison, A/B logs |
| 5 | Re-enable monitoring & alerts | ML Ops | Prometheus/Grafana/Evidently |
| 6 | Communicate rollback to stakeholders | Tech Lead | Slack/email/status page |

> ? Time to rollback: Target < 15 minutes (via feature flag or canary rollback).

? 3. Rollback Triggers (Automated or Manual)

Set up automated alerts to detect when rollback should start:

| Trigger | Threshold | Detection Method |
|-------------------------------|---------------------------------------|------------------------------|
| Online precision < 0.75 | 24-hour average | Business metric dashboard |
| Prediction distribution shift | PSI > 0.2 on output scores | Drift monitoring (Evidently) |
| Feature drift on `plan_tier` | PSI > 0.15 or KL > 0.3 | Feature monitoring |
| Hash collision rate > 5% | Simulate hash on live values | Logging + post-hoc analysis |
| Prediction drift per segment | Enterprise vs. Free precision gap > 1 | |

Engineering counterpart
7:50 AM - Responded in 0s

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

Agent response

7:51 AM - Responded in 57s

Absolutely. Here's a tight, action-oriented plan with concrete next steps, clear owners, and realistic timing - designed for immediate execution by an ML team.

? Incident Response Plan: Train/Serve Skew in churn_classifier_v3

Goal: Restore precision, eliminate skew, and prevent recurrence.

? Next Steps (48-Hour Remediation Plan)

| Step | Action | Owner | Timing | Deliverable |
|------|---|-------------------|------------|--|
| 1 | Declare incident & activate response team | Tech Lead | Now (T+0h) | Incident ticket (e.g., Jira, PagerDuty) |
| 2 | Rollback inference encoding to one-hot via feature flag | ML Engineer | T+0-2h | Serving path restored; log verification |
| 3 | Verify rollback success:
- Check prediction logs
- Confirm one-hot columns in input tensor
- Compare to baseline precision | ML Engineer / SRE | T+2-4h | Verification report |
| 4 | Re-enable monitoring:
- Precision metric alert (threshold: <0.75)
- Feature drift on `plan_tier`
- Prediction drift (PSI) | ML Ops | T+4-6h | Alerts active in Grafana/Evidently |
| 5 | Freeze model retraining until pipeline is aligned | ML Lead | T+0-24h | Policy: "No training without serving parity" |
| 6 | Reproduce issue in staging:
- Deploy hash encoding path
- Replay 1 week of traffic
- Measure precision drop & collisions | ML Engineer | T+6-24h | Debug report with hash collision rate |
| 7 | Quantify hash collision risk:
- Map all `plan_tier` values to hash buckets
- Log collisions across all categories | Data Engineer | T+12-24h | Collision audit log |
| 8 | Design aligned pipeline:
- Option A: Retrain model using hash encoding (if needed)
- Option B: Use embedding or ordinal encoding
- Choose by T+48h | ML Architect | T+24-48h | RFC: "plan_tier Encoding Strategy" |

? Rollback Status & Validation Criteria

| Criteria | Expected Outcome | Owner | Status |
|--|---|----------------|--------------------------|
| ? Serving uses one-hot encoding | Logs show `plan_tier_free`, etc. | ML Engineer | ☐ |
| ? Online precision ? 0.78 within 24h post-rollback | Metric recovers to baseline | Data Scientist | ☐ |
| ? No new hash-encoded predictions in logs | No integer bucket values in model input | SRE | ☐ |
| ? Drift alerts re-enabled | PSI/prediction alerts firing in staging | ML Ops | ☐ |

> ? Rollback complete when all boxes checked.

? Team Roles & Responsibilities

| Role | Owner | Responsibility |
|----------------|-------------|-------------------------------------|
| Tech Lead | Jane Doe | Incident command, comms, escalation |
| ML Engineer | John Smith | Rollback, logging, staging test |
| ML Ops | Alex Lin | Monitoring, rollback deployment |
| Data Scientist | Sam Chen | Metric validation, slice analysis |
| Data Engineer | Taylor Reed | Hash audit, schema logging |

? Timeline Summary

| Timeframe Milestone |
|-----------------------|
| ----- ----- |
| Now (T+0h) Incident |

Engineering counterpart
7:51 AM - Responded in 0s

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

Agent response

7:52 AM - Responded in 16s

The hash encoding implementation and its collision behavior - that's the one thing I'd urgently want a second pair of eyes on.

? Why?

Because it's invisible, non-intuitive, and irreversible - and in this case, it's the linchpin of the train/serve skew.

Even a small hash collision (e.g., "enterprise" and "free" mapping to the same bucket) can:

- Destroy signal for high-value segments
- Cause systematic false positives (hurting precision)
- Be completely undetectable without inspecting the raw hash logic

? What a Second Set of Eyes Should Review:

| Focus Area | What to Check |

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

| Hash function | Is it deterministic? (e.g., `hashlib.sha256`, not `Python hash()` which salts) |

| Bucket size | Is it too small? (e.g., `% 4` -> 25% collision chance even with perfect hash) |

| Collision audit | Manually test:

`hash("free") % N`, `hash("pro") % N`, `hash("enterprise") % N` - do any collide? |

| Consistency across services | Is the same function used in training (if retraining) and serving? |

| Unknown inputs | How is a new tier (e.g., "premium") handled? Same bucket as existing? |

? Example: A Silent Collision

```
```python
```

```
In production
```

```
hash("enterprise-user") % 3 -> 1
```

```
hash("free-trial") % 3 -> 1 <- collision!
```

```
```
```

-> Model sees identical input for two very different users -> treats enterprise like free -> mispredicts churn -> precision drops.

This would not show up in logs unless someone explicitly checks the hash output.

? Request to Colleague:

> "Can you independently validate the hash logic in the inference service?

> Specifically: run the three current `plan\_tier` values through the actual encoder and confirm:

> 1. The function is deterministic

> 2. No collisions occur

> 3. The bucket size is large enough (ideally ? number of categories)

>

> Bonus: simulate what happens if a new tier like 'premium' appears."

? Bottom Line:

> You can't trust a hash encoding you haven't seen and tested - especially when performance dropped right after it went live.

A second pair of eyes here isn't just helpful - it's a risk mitigation necessity.

Engineering counterpart**7:52 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.