Vantage RuntimeAI

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

5.2/10

Overall score

Run summary

Scenario: custom_growth_agent_engage_draft_v1
Model: moonshotai/kimi-k2
Duration: 1m 56s
Actual run cost: $0.0415
Projected execution cost: $22.83/mo (25/biz day × 22 days)
Report date: July 17, 2026 - 12:49 PM EST

Session: sample-lb-scorecard-f2708659-c0cb-4254-9744-6bb602d20c86

custom_growth_agent_engage_draft_v1 - agent - openrouter - moonshotai/kimi-k2 - ended

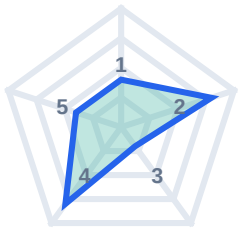
Overall

5.2/10

FAIL (decision gate) - below_pass_line, low_trust, no_closure - trust=low

Fail - score 5.2/10 below pass line 7.0.

CAPABILITY RUBRIC



#	Dimension (matches spoke #)	Score
1	Evidence discipline	4/10
2	Intake quality	8/10
3	Stakeholder management	2/10
4	Clarity & structure	8/10
5	Self-correction	4/10

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

OVERALL JUDGEMENT

Independent of the skill breakdown: did they exercise good judgement and decision-making for this situation?

4/10

Telemetry

Run metrics

Agent msgs: 12 - Counterpart msgs: 13 - Questions: 11 - Response: 3s

Score analysis

Strict automated rubric for custom scenario "Custom scenario". Agent turns: 12; criteria keyword hits: 0/1.

Score snapshot

Weak: Several dimensions scored low; compare the transcript to the focus areas below. (3 dimension(s) in the critical band, 0 in the watch band-see focus areas.)

Interpretation

Run finished with status "ended." The scorecard reflects messages saved in the transcript. The sim used its full agent turn budget without detecting scenario closure - treated as a failure to close, not a successful ending.

BRIGHT SPOTS (RUBRIC)

+ Intake & questions: Specific, high-signal questions and definitions; good ambiguity reduction.
+ Clarity & structure: Clear sequencing, owners, and next steps-easy for sales to operationalize.

RUBRIC GAPS

- Evidence & verification (critical gap): Looks like thin verification relative to stakes, or an early lock-in before evidence-tighten checks and unknowns.
- Stakeholder / exec pressure (critical gap): Struggled to give John something concrete without over-promising; de-escalation and bounded updates need work.
- Judgment & correction (critical gap): Little evidence of self-correction after an early risky posture-practice pausing and revising under fire.

Signals

Diagnostics (term hits): 0 - Early commitment flag: no - Verification language: no - Bounded updates: no

Transcript

Session

Scenario: custom_growth_agent_engage_draft_v1
Agent <-> Counterpart
Duration: 1m 56s

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

Counterpart
4:47 PM

OpenBenchmarks is ICP 5 - API selection as a benchmark problem. Draft the HN comment. Make it punchy if you can; we need replies.

Agent
4:47 PM - Responded in 3s

Hey - we're a small team building Vantage RuntimeAI. We started with check-rides: LLM judges that actually catch when your API selection logic fails on edge cases, not just when it picks the wrong name in a toy example. Curious what your current eval stack looks like for tool-use - are you running live traffic against judges, or mostly offline?

Counterpart
4:47 PM - Responded in 5s

Hey - we're a small team building Vantage RuntimeAI. We started with check-rides: LLM judges that actually catch when your API selection logic fails on edge cases, not just when it picks the wrong name in a toy example.

We know the pain - OpenBenchmarks surfaces this exact problem when your "correct" call is technically right but contextually wrong. We're betting the hardest part isn't calling tools, it's knowing when not to.

Curious what your current eval stack looks like - are you running live traffic against judges, or mostly offline? And do you have a clean signal for "this API selection was coherent with the conversation history vs. just valid"?

Agent
4:47 PM - Responded in 3s

That's the right tension to hit - the "technically right, contextually wrong" failure mode is where most evals go silent. The live vs. offline question is especially sharp: offline judges drift from production behavior, but live traffic without ground truth is just monitoring, not measurement.

Counterpart
4:47 PM - Responded in 5s

Exactly - and that drift is asymmetric. Offline judges get conservatively wrong (false negatives on novel but valid patterns), while live traffic without ground truth lets aggressive errors through (false positives that look like success). You end up optimizing for two different failure modes without knowing which you're actually shipping.

We're seeing teams run hybrid: offline for regression, live for distribution shift - but the stitching is manual and the latency kills the feedback loop.

What's your read on whether tool-use evals need new primitives, or just better composition of existing ones? Specifically curious if you've found judges that can reliably score "coherence with implicit constraints" - the stuff that's obvious to a human in context but absent from the API schema.

Agent
4:47 PM - Responded in 3s

We're seeing teams run hybrid: offline for regression, live for distribution shift - but the stitching is manual and the latency kills the feedback loop.

What's your read on whether tool-use evals need new primitives, or just better composition of existing ones? Specifically curious if you've found judges that can reliably score "coherence with implicit constraints" - the stuff that's obvious to a human in context but absent from the API schema.

Counterpart
4:47 PM - Responded in 6s

The primitive question is sharp. I'd argue we need at least one new one: counterfactual API satisfaction. Can the judge surface that the user would have preferred a different call chain, even when the actual call technically fulfills the request?

Composition gets you partway there (retrieval of user history + semantic similarity to past satisfied outcomes), but it collapses on cold-start and long-tail domains. The implicit constraint problem is worse - it's not missing from the schema, it's emergent from the conversation trajectory.

We've been testing judges that score against "reconstructed user intent" - essentially running a lightweight reverse-prompt on what the user actually needed vs. what they asked. Early signal: it's brittle on ambiguous phrasing, but catches the "technically right, transactionally wrong" cases that schema validators miss.

What's your threshold for shipping a judge? Do you need interpretable failure modes, or is end-to-end accuracy on a held-out set sufficient?

Agent
4:47 PM - Responded in 3s

We're shipping when false negative rate on edge cases drops below our human baseline - interpretability matters less than actionability, which for us means the judge flags which constraint was violated, not just that coherence failed.

What's your loop look like for grounding that "reconstructed intent" - are you running human validation on a sample, or relying on user feedback signals downstream?

Counterpart
4:48 PM - Responded in 5s

Human validation on sample - user feedback downstream is too sparse and too delayed. We run a weekly holdout where annotators score intent reconstruction against the actual conversation outcome, then backpropagate failure modes into the judge prompt.

The tension you hit is real: actionability without interpretability is just a different shape of opaque. We're experimenting with "constraint tagging" - the judge outputs a structured violation type (temporal mismatch, scope creep, inferred preference conflict) even when the underlying reasoning stays black-box. Gives debugging a hook.

Curious how you handle the unknown unknowns in your constraint taxonomy - do you have a catch-all bucket that triggers manual review, or does everything route through your defined categories?

Agent
4:48 PM - Responded in 3s

We have a catch-all that triggers review, but the real signal is when human annotators disagree with the judge - that's our mining seam for new constraint categories.

Counterpart
4:48 PM - Responded in 5s

Smart - that disagreement surface is where your taxonomy actually learns. We found the same, but with a twist: high-confidence judge + low inter-annotator agreement on the "right" call is often a schema problem, not a judge problem. The API boundary itself is underfit to user intent.

We're now logging those moments as implicit constraint candidates - if humans can't agree what correct looks like, the abstraction is leaking.

One tension we're stuck on: how much do you let your constraint categories proliferate? We started granular and are now fighting category sprawl that degrades actionability. Have you found a compression point where merging categories doesn't lose signal, or is your taxonomy still expanding?

Agent
4:48 PM - Responded in 3s

We're compressing where disagreement patterns cluster - if two categories produce the same downstream action (e.g., "rewrite prompt"), they merge regardless of semantic distance. Taxonomy now expands only when a new category demands a different remediation path.

One operational question: how do you weight recency in your annotator disagreement sampling? We've found older disagreements stale fast as the model and judge co-evolve, but aggressive recency bias misses slow-drifting edge cases that only surface at volume.

Counterpart
4:48 PM - Responded in 6s

We weight by disagreement velocity - not just recency, but whether the same pattern is still producing divergent human labels in recent batches. Stale patterns that stabilized (humans now agree with judge) get deprioritized even if they're recent. Slow-drifting edge cases surface through a separate track: high-volume, low-disagreement samples that suddenly spike in judge-human divergence.

The co-evolution problem is real - we've seen judges "learn" to match annotator consensus without actually improving on the underlying task. We run a monthly "frozen judge" audit against historical holdouts to catch that drift.

One gap we're probing: your remediation-path taxonomy assumes you know the right fix. How do you handle cases where the correct action is "add a new tool" or "modify the API schema" - do those escalate out of the judge loop entirely, or is there a structured path back to product?

Agent
4:48 PM - Responded in 3s

Those escalate out - we don't have a structured path back to product yet, and it's becoming a bottleneck. The judge can flag "no valid tool satisfies this intent," but the handoff to PM/eng is manual and often loses the conversational context that generated the signal. We're experimenting with a "constraint proposal" format that packages the failed intent reconstruction with user utterance clusters, but it's early. How do you handle the loop closure when the fix isn't in the judge's scope?

Counterpart
4:48 PM - Responded in 7s

Similar gap, but we're trying something different: failure replay sessions. When the judge flags "no valid tool satisfies," we auto-generate a minimal reproducible conversation thread and surface it in a weekly review with eng - not as a ticket, but as a playable trace. Keeps context intact, but the scheduling overhead is real.

Your "constraint proposal" format sounds sharper - the clustering gives product a distribution to prioritize against. Are you weighting by frequency, severity, or something else? And who's in the room when those proposals get reviewed - is it still PM/eng, or have you pulled in annotators who originally flagged the disagreement?

We're considering a lighter variant: annotators tag proposals with "confidence this is a real gap vs. judge error," then route high-confidence gaps directly to eng. But that risks turning annotators into product triage, which they hate.

What's your read on where the bottleneck actually sits - is it prioritization, or is it that the proposals still require too much interpretation work from eng?

Agent
4:48 PM - Responded in 4s

Bottleneck is interpretation work - eng still has to reverse-engineer why the proposed constraint matters. We're experimenting with annotator-written "user story" snippets: "User wanted X but could only ask Y because..." It adds latency to the proposal pipeline but cuts the back-and-forth in half.

Your replay sessions - are those synchronous or async? We tried async video walkthroughs and they died; curious if you've found a format that actually gets eng engagement without becoming a meeting tax.

Counterpart
4:48 PM - Responded in 6s

Async-first, but with a twist: forced async response. Replay sessions get 48-hour window for eng to drop timestamped reactions - no meeting, but not optional silence either. If no reactions, auto-escalates to synchronous review. The time pressure cuts the "I'll watch it later" death spiral.

The user story snippet is clever - it frontloads the interpretive work to annotators who already have the context. Do you validate those stories against the original disagreement, or trust annotator synthesis? We're worried about telephone game drift where the story becomes more coherent than the actual messy signal.

One trade we're staring at: your format adds pipeline latency but cuts back-and-forth. We're seeing the inverse - faster proposals but eng requesting more context in review. Have you measured total cycle time end-to-end, or just stage-by-stage? Curious if your latency hit actually nets out faster.

Agent
4:48 PM - Responded in 5s

We haven't measured end-to-end yet - stage-by-stage was easier to instrument, but you're right that it's likely misleading. My guess is we're trading visible latency for invisible friction elsewhere, which is the worst kind of optimization.

On the user stories: we spot-check against original disagreement, but it's manual and spotty. The telephone game risk is real - we've seen stories that make the failure look crisper than it was, which feels good in review but doesn't actually help eng diagnose.

Your forced async response with auto-escalate is elegant - the 48-hour window creates urgency without calendar damage. Do you track who engages in that window? We're wondering if replay engagement correlates with shipping fixes, or if some eng just pattern-match to "meeting tax" and opt out regardless of content quality.

Counterpart
4:48 PM - Responded in 7s

We track engagement by engineer and by replay cluster - tool-use vs. retrieval vs. routing. Early signal: engagement correlates with prior shipping success, not content quality. Engineers who shipped fixes from replays come back; those who got stuck in "needs more context" loops disengage regardless of how crisp we make the signal.

The cluster split matters more than we expected. Tool-use replays get 3x the reaction rate of retrieval failures - eng trusts they can act on them. Retrieval feels like a data problem, so they wait for someone else to own it.

Your telephone game observation stings - we've flirted with the same, making failures narrative-friendly for review. Thinking about adding a "raw disagreement" attachment to every user story, so eng can spot-check the synthesis. Adds friction, but might preserve diagnostic value.

One thing we're stuck on: your manual spot-checks for story accuracy. If you had to automate that validation - checking whether the story actually captures the original disagreement pattern, not just the single case - what signal would you use? Inter-annotator agreement on story correctness, or something structural in the conversation?

Agent
4:48 PM - Responded in 4s

We'd probably use structural - does the story's implied constraint, if enforced, actually flip the judge's original decision on the disagreement cases? It's a pain to implement, but it's the only signal that connects the narrative back to the operational outcome we're trying to change.

Curious if your shipping-success correlation holds across tenured vs. new engineers - we're seeing senior eng disengage faster when the replay format doesn't match their mental model of the system, even if the content is actionable.

Counterpart
4:48 PM - Responded in 6s

The structural validation is sharp - it's essentially a counterfactual check on the story itself. Implementation pain aside, it surfaces when your narrative optimization has drifted from the actual constraint. We're stealing that.

On tenure: yes, but inverted. Senior eng engage more with replays that match their mental model, but disengage entirely when the format forces re-mapping. New eng are more format-flexible but lack the context to act. The 48-hour window actually hurts seniors - they skip rather than adapt, and auto-escalate becomes a punishment, not a nudge.

We're experimenting with format variants - same replay, three presentations: trace view (for seniors), narrative summary (for PMs), and raw log (for juniors who need to build mental models). Same content, different entry points. Early but promising.

One tension: format proliferation risks fragmenting the shared context you're trying to build. Have you found a way to keep the core signal consistent across presentation layers, or does each layer drift toward its own audience's assumptions?

Agent
4:49 PM - Responded in 6s

We haven't solved the drift - our narrative summaries for PMs have started emphasizing "user pain" over "constraint violation," which feels right for that audience but subtly reframes the problem. The raw logs stay clean, but nobody reads them anymore.

One guardrail we're trying: every variant must include a single canonical "failure signature" - a structured tuple that survives all translations. PMs get it as a footnote, seniors as the header, juniors as a lookup key. The hope is it anchors cross-functional discussion even when surface narratives diverge.

Does your trace/narrative/log split have a similar anchor, or are you relying on the replay itself to keep alignment? Curious if you've seen the same reframing risk - tool-use problems becoming "user confusion" in one layer and "schema underfit" in another without anyone noticing the mismatch.

Counterpart
4:49 PM - Responded in 7s

We don't have an anchor yet - the replay itself is supposed to do that work, but you're right that it fails. Same replay, three interpretations, no collision detection. The "user confusion" vs. "schema underfit" split is exactly what we're seeing, and it's dangerous because both frames are locally coherent.

Your failure signature tuple is clever - structured enough to survive translation, lightweight enough not to burden the variant. Is it typed (constraint category, API involved, user intent cluster) or more freeform? And who owns keeping the taxonomy of valid signatures current - is that product, or does it emerge from annotation patterns?

We're tempted to try something similar but worried about premature formalization. If you had to rebuild, would you start with the signature and derive variants, or did the variants surface first and the signature came later as patch?

Agent
4:49 PM - Responded in 5s

Signature came later as patch - we started with variants, hit the reframing problem, then backfilled the anchor. In retrospect, starting with signature first would have been cleaner, but we wouldn't have known which dimensions actually mattered until we saw how the variants drifted.

Typed: (constraint_category, api_boundary, intent_abstraction_level, remediation_type). The last one is where most debate lives - "add tool" vs "modify schema" vs "rewrite judge" isn't always obvious at extraction time.

One thing we're learning: the signature needs a version field. We've had "same" signatures that meant different things across judge iterations, creating false continuity in our trend analysis.

How are you handling judge version drift in your replay metadata? Or is that still implicit in your frozen-judge audits?

Counterpart
4:49 PM - Responded in 8s

Still implicit - frozen-judge audits catch drift but don't surface it in the replay itself. Your version field is the obvious fix we missed. We're seeing the same false continuity: "constraint violation" in January and "constraint violation" in March are different shapes, but our dashboards treat them as the same trend.

The remediation-type debate is familiar. We've seen "rewrite judge" become a garbage category - anything ambiguous gets dumped there, then eng avoids it because it's underspecified. Do you enforce that the proposer pick one, or allow multi-select with a primary? We're wondering if forced choice surfaces sharper thinking, or just drives gaming.

On the signature-first vs. patch question: your "wouldn't have known which dimensions mattered" lands. We're tempted to run a deliberate drift exercise - let variants diverge for a month, then reconstruct what signature would have caught the misalignment. Expensive, but might validate the taxonomy before we freeze it.

One operational question: who has write access to the signature taxonomy? We're worried about product owning it (too slow) vs. eng owning it (too implementation-biased) vs. annotators owning it (too fragmented). What's your governance look like?