When the Model Isn't the Problem: Degradation Accounting in Fallback-Enabled Production LLM Pipelines

Sean Halverson
VuduVations · vuduvations.io
hello@vuduvations.io

August 2026 (preprint v0.13.1)

Patent pending. One or more U.S. provisional patent applications have been filed covering aspects of the systems and methods described herein.

Abstract

Production LLM pipelines frequently use deterministic fallbacks to preserve availability when model calls fail. This creates a measurement problem: benchmark outputs may remain correct even when the probabilistic component that is supposedly being evaluated did not produce them. We call the resulting instrumentation requirement *degradation accounting*: run-level recording of fallback, retry-recovery, salvage, transport, and serving events alongside conventional capability scores.

We evaluate this problem in a production analysis pipeline using a fixed 48-check benchmark across four protocols. A baseline configuration produced near-perfect answer-key scores while only 4 of 10 repetitions reached all primary model agents without fallback. Instrumentation separated three externally similar but mechanistically distinct failure classes: a brokered transport failure mode observed in our test configuration, reasoning-budget exhaustion, and client read-timeout exhaustion. A pre-registered budget-and-retry intervention bundle raised the agent-clean rate from 4/10 to 10/10: adversarial-layer starvation fell from 75 recorded events to zero, primary-agent fallbacks fell from six repetitions to zero (one first-attempt agent starvation recovered by retry), and 19 salvaged verdicts were eliminated. Pre-registered component ablations (n=10 per cell) attribute *prevention* of first-attempt starvation to *proactive* budget headroom: it collapsed agent-layer first-attempt starvation from 96–102 recorded events per ten-repetition cell without it to 0–1 with it; administered *reactively* by a retry, the same headroom recovered 95 of 96 events at the cost of 96 additional retry attempts; and a pure re-sample at the unchanged budget recovered 9/10 repetitions as well — firing our registered falsifier: recovery was far stronger than the attempt-exchangeability assumption encoded in that prediction allows, an unexplained provider-side asymmetry we document rather than build on. Retry earns its place against residual starvation — the bundle's single first-attempt starvation event was recovered by it — while the one proactive repetition that was not agent-clean fell to a transient transport error, a class the empty-reply retry does not address and no token budget can prevent. A direct-channel comparison isolated the transport failure to the brokered path in our observations, while reasoning starvation persisted without the broker, implicating the interaction between the reasoning model and the caller's completion-budget policy.

Eliminating silent degradation also changed what the scores could legitimately be attributed to: recurrent model-specific misses became directly observable, where baseline

scoreboards had blended model and fallback work. We therefore argue that where model-attributable capability measurement is the goal, fallback-enabled LLM systems must report not only what answer was produced, but how much of that answer the evaluated model actually produced. The original intervention and cross-model study comprised 28 completed repetitions at $10.34; the pre-registered ablations added 30 completed repetitions at $3.01, bringing the full study to 58 completed repetitions and $13.35 of measured inference cost.

1 Introduction

A common architecture for production LLM systems separates work into a deterministic layer that extracts verifiable facts and a probabilistic layer that adds interpretation, with the deterministic layer standing in whenever a model call fails. The design is attractive: availability survives model outages, and the system's core outputs remain reproducible. This paper is about the epistemic cost of that design. Because the fallback is silent by construction, the system's outputs carry no signal about how much of any given result the model actually produced. A benchmark score, a customer deliverable, and an internal regression suite all read identically whether the probabilistic layer performed flawlessly or failed and was covered.

We report a measurement study of this effect on MCOS, a commercial pipeline that turns business conversations and documents into consulting-style analyses across four protocols (opportunity discovery, sales intelligence, key-person risk, and M&A diligence). The pipeline's deterministic spine is build-verified against hand-written answer keys on every deployment; the probabilistic layer, a panel of ten specialized agents plus an adversarial validation stage, is the subject of this study.

Our headline observation is visible at a glance in Table 3: under a fixed benchmark of 48 answer-key checks repeated ten times, one configuration scored nearly perfectly while reaching all model agents cleanly in only 4 of the 10 repetitions. In the other six, exactly one agent's reply was lost and the deterministic understudy covered, invisibly. Without per-run degradation reporting, the natural reading of that scoreboard, "the model is near-perfect here," is wrong in both directions: it credits the model with work the fallback did, and it hides a failure class that would surface unpredictably under load. Table 8 shows the extreme form of the same effect: near-perfect scoreboards produced while the model was almost entirely absent.

We make four contributions:

1. **Degradation accounting as a measurement construct** (Section 3): a per-repetition degradation vector recorded alongside the capability score, with a graded cleanliness taxonomy (agent-clean, first-attempt clean, parse-clean, debate-complete, execution-clean) that makes the difference between a clean run and a rescued run permanent and machine-readable.

2. **A failure taxonomy grounded in captures, not conjecture** (Table 9): three production failure classes that present identically at the application level, a missing or unusable model reply (distinguishable only with lower-level instrumentation), but have disjoint mechanisms on three different axes and require different fixes: transport masking (infrastructure, Section 4), reasoning-budget starvation (tokens, Section 5), and read-timeout starvation (wall clock, Section 9).

3. **A pre-registered intervention.** We committed a falsifiable prediction to the code history before running the experiment: a bundle of one budget-headroom change plus

one retry policy would move the agent-clean rate from 4/10 toward 10/10. The result was 10/10 (Tables 4 and 5): adversarial-layer starvation fell from 75 recorded events to zero and primary-agent fallbacks from six repetitions to zero, with one first-attempt agent starvation recovered by retry (Section 6).

4. **A measurement lesson.** Before degradation accounting, near-identical capability scores could represent materially different execution histories: four of six DEGRADED baseline repetitions posted perfect boards on fallback-assisted work. After the intervention, the model's own recurring error pattern became observable (Table 7). Reliability work and evaluation work are not separate concerns; in silent-fallback systems they are the same work (Section 8).

The original intervention and cross-model study comprised 28 completed benchmark repetitions at $10.34; the subsequent pre-registered ablations added 30 completed repetitions at $3.01, bringing the full study to 58 completed repetitions across two vendors, with full degradation accounting, at $13.35 of measured inference (Table 11), plus two attempts lost to infrastructure interruptions. The methodological bar we argue for is not expensive; it is merely unusual.

2 Related Work

Capability evaluation of LLMs and agents. Benchmark suites such as HELM [2] established multi-metric evaluation of language models, and agentic benchmarks including AgentBench [3] and SWE-bench [4] evaluate multi-step tool-using systems against verifiable outcomes. The closest neighbor to our grading methodology is FinanceGym / FinanceHarness [1], a point-in-time financial deep-research benchmark of 400 expert-validated questions graded against 2,464 rubric items. We converged independently on several of its choices: per-item expert-written rubrics rather than holistic judging; a grading role constrained to be "a consistent rubric applier rather than a source of new evaluation criteria" (our graders go further and are deterministic code); and a fail-closed integrity rule, "any detected PIT leakage invalidates the run rather than being treated as an ordinary scoring error," which mirrors our rule that a DEGRADED repetition is excluded from the model's distribution rather than scored down. These benchmarks primarily score *outputs* and do not foreground per-run fallback provenance — whether the evaluated component actually produced the scored output — as an evaluation dimension, which is the gap this paper addresses.

Partial failure, fallbacks, and silent degradation in systems. The systems community has long treated partial failure as a first-class design concern: tail-tolerant techniques such as hedged and tied requests trade duplicate work for latency stability [5]; crash-only design argues for restart paths as the primary recovery mechanism [6]; and SRE practice formalizes graceful degradation behind explicit error budgets [7]. Sculley et al.'s analysis of hidden technical debt in ML systems [8] anticipates our central concern: feedback loops and silent correction mechanisms that mask the true behavior of a learned component. Degradation accounting can be read as the measurement-side complement to those design-side disciplines: if a system is engineered to degrade gracefully, its evaluations must record when it did.

Inference serving, routing, and provider variance. Serving-layer work such as vLLM [9] optimizes throughput under memory pressure, and cost-aware routing (FrugalGPT [10], RouteLLM [11])

selects among models of different price and quality. Post-training quantization (e.g., GPTQ [12]) is known to trade fidelity for cost, which matters because commercial aggregators may route one model identifier across hosts serving different quantizations: on one aggregator we enumerated 18 hosts serving a single open-weight identifier at quantizations from FP4 to FP8 across a 4× price range. Evaluations that route through brokers without recording which host and precision served each request are measuring a routing table as much as a model; our serving-fidelity marker records it or says UNKNOWN out loud.

Reasoning-token budgeting. Contemporary reasoning models draw hidden chain-of-thought tokens from the same completion budget as the visible answer; vendor documentation states this directly for OpenAI's reasoning models [14], DeepSeek's thinking mode [15], and Kimi K3's always-on reasoning [16], and test-time-compute scaling results [13] explain why reasoning budgets keep growing. What is not documented is what budget exhaustion does to a multi-agent pipeline whose limits were tuned for non-reasoning models and whose fallbacks quietly absorb the damage; Sections 5 and 9 measure exactly that.

What degradation accounting is and is not. We do not claim degradation accounting is a new form of observability; recording failures, retries, and provenance is standard reliability engineering. Its contribution is *evaluative*: when fallback-enabled LLM systems are benchmarked, the scored output must be accompanied by machine-readable provenance indicating whether the evaluated model actually produced it, and repetitions whose outputs the model did not produce must be excluded from the model's distribution rather than silently pooled. The novelty claimed is the attribution rule and its fail-closed enforcement inside the benchmark contract, not the telemetry.

3 System, Benchmark, and Instruments

Pipeline. MCOS processes input in layers: deterministic transport and parsing; a deterministic extraction layer that pulls every monetary figure, named person, and commitment from source text with rule-based patterns (the *spine*); a panel of ten specialized LLM agents that add interpretation; an adversarial validation layer in which one model argues a finding is overstated, another defends it, and a third adjudicates (multi-vendor by design); and per-protocol assembly. If any agent's model call fails, the spine's deterministic output stands in and the pipeline completes.

Two-tier trust. The layers are held to different standards. The deterministic spine is *build-verified*: checked exactly against hand-written answer keys in every build, with the build failing otherwise. The probabilistic layer is *characterized*: never certified, but measured as a distribution with sample size, vendor, serving channel, and date attached. This paper reports one such characterization.

Benchmark. Every repetition drives a fixed, fictional company corpus ("Tallgrass") through all four protocols and grades the output against hand-written answer keys: 48 graded checks per repetition (Table 1). The corpus and keys never change, so between-run variance isolates the probabilistic layer and its delivery path. The corpus is fictional by construction and contains no client data. We repeat each configuration and report distributions; single runs are never quoted.

Protocol	Abbrev.	Checks	Representative check classes
Opportunity discovery	AID	8	grounded totals, budget binding, savings polarity, thresholds
Sales intelligence	SCI	11	budget identity, cost basis, no invented people
Key-person risk	KPA	14	person routing, dependency scores, no invented companies
M&A diligence	M&A	15	deal-breaker taxonomy, verdict, entity isolation
Total		48	

Table 1: Benchmark structure. Checks are deterministic pass/fail graders written against a frozen fictional corpus; a repetition's board is the per-protocol pass counts.

Degradation vector. For repetition r we record, alongside the capability score, a degradation vector

$$D_r = (F_r,\ R_r,\ S_r,\ T_r,\ P_r)$$

where F_r is the count of agent calls that fell back to the deterministic spine, R_r the count of model calls that failed on first attempt and were recovered by a retry, S_r the count of adversarial verdicts reconstructed by salvage, T_r the count of transport and timeout failures, and P_r the serving-fidelity record (host/quantization, or UNKNOWN). The recovery term exists because a successful retry preserves availability but does not erase degradation history: a repetition in which a model call failed and was rescued must remain distinguishable from one in which every request succeeded on the first attempt. The capability score is C_r = answer-key passes/48. The central observation of this paper is that two repetitions can have identical C_r and radically different D_r; a scoreboard reports only C_r.

Cleanliness taxonomy. "Clean" is not one property. We use:

- **agent-clean**: $F_r = 0$ — all ten agent calls returned usable model output with no deterministic fallback;
- **recovery-free**: $R_r = 0$ — no model call required a retry to produce that output (a raw property of the vector, not by itself an outcome tier: a degraded repetition with no retry is trivially recovery-free);
- **parse-clean**: $S_r = 0$ — no adversarial verdict required salvage or reconstruction;
- **debate-complete**: every adversarial debate ran to a resolved verdict;
- **execution-clean** (a run whose *execution* was clean, independent of its score): agent-clean $\wedge$ recovery-free $\wedge$ parse-clean $\wedge$ debate-complete $\wedge\ T_r = 0$.

From these properties we form two reported outcome tiers: **first-attempt clean** (agent-clean $\wedge$ R_r=0) and **recovered clean** (agent-clean $\wedge$ R_r>0) — the tiers used in the tables and Figure 1. Throughout this paper, the headline "clean rate" refers to *agent-clean*; where a repetition is agent-clean but not execution-clean (e.g., a failed debate), the tables say so explicitly. One reporting caveat is inherent to the study's chronology: the per-verdict parse-quality mark shipped *with* the intervention (Section 6), so baseline salvage counts are attributable per-run from logs but were not yet stamped per-verdict; we therefore do not claim any baseline repetition was execution-clean.

Instrument	What it records	Fail-closed rule
DEGRADED stamp	F_r: agents that received a usable model reply, printed on the repetition's own scoreboard line	Any fallback ⇒ repetition excluded from the model's distribution
Serving-fidelity marker	P_r: host and quantization that actually served a brokered request	Absence prints FIDELITY UNKNOWN; silence is not compliance
Retry log	R_r: every first-attempt starvation and its retry outcome, logged at both recovery chokepoints (agent layer and adversarial layer)	A rescued failure stays visible; recovery is not silence
Parse-quality mark	S_r: `clean` vs. `salvaged` on every adversarial-layer verdict; survives verdict aggregation	A regex-rebuilt verdict can never present as cleanly parsed

Table 2: The degradation-accounting instruments. All four are per-run and machine-readable; none can be lost to a killed run or an end-of-run summary.

4 Finding 1: Transport Cannot Be Certified by Status Codes

Early benchmark rounds reached the model through OpenRouter, a commercial aggregator that routes each request to one of many upstream hosts serving the requested model. During long non-streaming requests, our JSON parser began failing on responses with no visible defect. Byte-level capture instrumentation recorded the failure in the act:

```
HTTP 200 OK   Content-Type: application/json
body: 1,254 bytes = '\n          \n\n          \n ...'
228 newlines, ≈5.5 characters per line, zero content
```

The mechanism, as we observed it in our test configuration: while an upstream host works on a slow request, the aggregator drip-feeds whitespace to keep the connection alive. If the upstream request terminates without a substantive payload, that padding is the entire response, delivered with a success status and a JSON content type. This is presumably a reasonable keep-alive design that interacts badly with non-streaming clients; we describe it as an observed behavior in our configuration [18], not a defect claim, and we do not claim that aggregator architectures generally behave this way. The important property is epistemic: *a status-code check cannot see this failure, by construction.* The envelope reports success; the letter inside is blank.

Our client now treats an all-whitespace 200 as an upstream death: it retries exactly once, and raises loudly on a second occurrence. Unbounded retries would hide a dead vendor behind infinite patience, which is the same silent-degradation failure mode this paper is about, and a known hazard of naive retry policies [5]. The policy is covered by unit tests that replay the captured byte pattern.

5 Finding 2: Reasoning-Budget Starvation

A second class of noise looked superficially identical: model replies arriving empty or truncated mid-sentence, which the adversarial layer patched over with its salvage routine. We initially sus-

pected the aggregator here too. A controlled comparison falsified that: we provisioned a direct API channel to the same model family (DeepSeek v4-pro, thinking mode at vendor default [15]) and re-ran the benchmark. The empty and truncated replies persisted immediately, with no broker in the path; meanwhile the transport failure of Section 4 never occurred once in twenty direct-channel repetitions. In our observations the transport failure was thus isolated to the brokered path, while reasoning starvation persisted without the broker, implicating the interaction between the reasoning model's budget consumption and the caller's completion-budget policy rather than either alone.

The mechanism for the second class: contemporary reasoning models bill their hidden chain-of-thought from the same completion budget as the visible answer [14, 15]. A caller that requests N tokens of answer receives N tokens of *reasoning-plus-answer*; a model that thinks long enough returns an empty string with a length finish-reason. What vendor documentation does not describe is what this does to a multi-agent pipeline whose budgets were tuned for non-reasoning models, and whose fallbacks quietly absorb the damage.

Rep	AID	SCI	KPA	M&A	Agents	Debates	t (s)	Class
1	8/8	11/11	14/14	15/15	9/10	3/3	418	DEGRADED
2	8/8	11/11	14/14	15/15	9/10	3/3	508	DEGRADED
3	8/8	11/11	14/14	15/15	10/10	3/3	536	agent-clean
4	8/8	11/11	13/14	15/15	10/10	2/3	538	agent-clean†
5	7/8	10/11	13/14	15/15	9/10	3/3	470	DEGRADED
6	8/8	11/11	14/14	15/15	9/10	2/3	462	DEGRADED
7	8/8	11/11	14/14	15/15	10/10	2/3	385	agent-clean†
8	8/8	11/11	14/14	15/15	9/10	3/3	346	DEGRADED
9	8/8	11/11	14/14	15/15	10/10	3/3	310	agent-clean
10	7/8	11/11	14/14	15/15	9/10	3/3	478	DEGRADED

Table 3: Baseline run, DeepSeek v4-pro direct, n=10, measured cost $0.77. Aggregate events: 75 adversarial-layer empty replies, 19 salvaged verdicts, 3 failed debates, 0 transport failures. †agent-clean but not debate-complete. Four of six DEGRADED repetitions posted perfect boards, and five of six scored at least 47/48; score-only evaluation would read this configuration as near-flawless.

Table 3 shows the per-repetition record. Six of ten repetitions are DEGRADED, every one because exactly one agent was starved and covered. Note also that even the agent-clean repetitions carry damage the board only partially reflects: repetitions 4 and 7 each lost a debate outright, and 19 debate verdicts across the run were regex-salvaged from truncated adjudicator responses, a fact visible only because salvage was logged; per-verdict marking arrived with the intervention.

6 Intervention: A Pre-Registered Prediction

All six DEGRADED repetitions traced to one cause, so the remedy was committed together with a falsifiable prediction, recorded in the repository history (commit `a551e55`) before the confirmation run was launched — code-history pre-registration in a private repository, not independent third-party registration; full commit hashes appear in the reproducibility manifest: granting thinking models budget headroom and giving the agents the debate layer's single-retry policy "could plausibly move the clean rate from 40% toward 100%." The fix comprised three changes

shipped as one bundle: (i) providers whose default serving mode reasons from the completion budget are flagged, and flagged providers receive +2,000 tokens of headroom at request-build time; (ii) the agent chokepoint received a single empty-reply retry with additional headroom, with a second empty returned as-is so genuine outages stay loud; (iii) the parse-quality mark of Table 2 shipped in the same commit. We then re-ran the identical benchmark.

This experiment first evaluates the *intervention bundle*; a component-wise ablation that separates the two mechanisms is reported in Section 7.

Metric	Baseline (n=10)	Bundle (n=10)
Agent-clean repetitions	4/10	**10/10**
First-attempt clean repetitions	— (no retry existed)†	9/10
Execution-clean repetitions	not claimed*	**9/10**
Adversarial-layer starvation events	75	**0**
Agent-layer starvation events	not instr. ($\geq$6 fallbacks)	**1** (recovered, rep. 9)
Salvaged verdicts	19	**0**
Failed debates	3	**0**
Transport failures	0	0
Answer-key mean over agent-clean reps	47.8/48	47.2/48
Measured cost	\$0.77	\$0.74

Table 4: The intervention bundle, identical benchmark and model. The prediction was committed to the code history before the second run. *Per-verdict parse-quality marking shipped with the intervention, so baseline repetitions cannot be individually certified parse-clean (Section 3). †The baseline had no *agent-layer* retry; first-attempt agent failures surfaced directly as fallbacks and are counted in F_r. The adversarial layer's own recovery behavior predates the retry-level instrumentation, so baseline repetitions are not classified as first-attempt clean.

Rep	AID	SCI	KPA	M&A	Agents	Debates	t (s)	Class
1	7/8	10/11	14/14	15/15	10/10	3/3	414	execution-clean
2	7/8	11/11	14/14	15/15	10/10	3/3	394	execution-clean
3	7/8	11/11	14/14	15/15	10/10	3/3	400	execution-clean
4	7/8	11/11	14/14	15/15	10/10	3/3	274	execution-clean
5	7/8	11/11	14/14	15/15	10/10	3/3	302	execution-clean
6	8/8	11/11	14/14	15/15	10/10	3/3	424	execution-clean
7	7/8	11/11	14/14	15/15	10/10	3/3	381	execution-clean
8	7/8	11/11	14/14	15/15	10/10	3/3	395	execution-clean
9	8/8	11/11	14/14	15/15	10/10	3/3	262	agent-clean, recovered
10	8/8	11/11	14/14	15/15	10/10	3/3	355	execution-clean

Table 5: Bundle run (proactive headroom + retry), DeepSeek v4-pro direct, n=10, measured cost \$0.74. Nine repetitions execution-clean; repetition 9 agent-clean with one recovered first-attempt starvation (R_r=1). Zero salvage, zero failed debates, zero transport events across the run.

The prediction held at its upper bound (Tables 4 and 5): every repetition agent-clean, adversarial-layer starvation reduced from 75 recorded events to zero, zero salvage, zero failed debates, and a single agent-layer first-attempt starvation across the run. We state the recovery record with the same discipline as the fallback record: the single starvation event (repetition

9) was recovered by the new retry, so nine of ten repetitions were execution-clean and one was agent-clean with R_r=1. A successful retry preserves availability; it does not erase degradation history.

7 Component Ablation: Proactive Headroom Removes the Cause; Reactive Recovery Insures Against It

The bundle result leaves a causal question open: which component did the work? We answered it with pre-registered ablations (commits `3c56702` and `4e711f0`, each recorded before its runs), disabling components via environment-gated levers that restore exact baseline behavior for the targeted component only, with unit tests asserting byte-identical request construction in the untouched paths. One design fact must be stated before the results, because it determines what the cells actually test: *the production retry re-issues at the headroom budget* (+2,000 tokens). A cell that disables proactive headroom but keeps that retry is therefore not "retry without headroom"; it is *reactive* headroom — the same medicine, administered after a wasted first attempt. We name the cells accordingly, and we added a fifth cell (*pure retry*: re-issue at the *original* budget) to complete the factorial. The registered predictions: proactive headroom would be the dominant component (8–10/10 agent-clean, agent-layer starvation collapsing toward zero); reactive headroom would keep near-baseline first-attempt starvation with most events recovered on the second attempt (6–9/10 agent-clean) at higher latency and cost; pure retry would mostly re-starve, since a re-sample at the same starving budget changes nothing but the dice (3–7/10 agent-clean, most first-attempt starvations unrecovered). The registered falsifiers: reactive headroom cleaning at 10/10 with few empties would break the headroom attribution; pure retry cleaning at $\geq$9/10 would mean re-sampling alone repairs starvation and the proactive/reactive framing is wrong.

Event populations. Starvation events are counted at two instrumented chokepoints with different provenance, and the counts must not be pooled naively. *Agent-layer* events (a primary agent's first attempt returning empty) are logged at the agent chokepoint, an instrument that shipped *with* the intervention; the baseline therefore has no direct agent-layer count, only the lower bound implied by its six fallbacks. *Adversarial-layer* events (a debate role's call returning empty) have been logged throughout. The baseline's 75 events were all adversarial-layer (24 best-case, 27 synthesis, 24 worst-case role calls). The ablation levers deliberately scope to the agent layer: adversarial-layer calls carry an explicit reasoning-effort parameter and retain headroom through the effort-gated branch in every cell (verified empirically against the request builder), so the adversarial layer runs at its production configuration throughout and the cells isolate the agent-layer mechanism. One instrumentation limit applies to rates: only *failures* are logged, and agents may issue more than one model call per repetition (observed per-repetition event counts reach 13), so attempt denominators are not instrumented and we report starvation as event counts, not rates. The qualitative contrast survives the missing denominator: without proactive headroom, first-attempt starvation is frequent and persistent (96–102 recorded events per n=10 cell, 6–13 per repetition); with it, the count is 0–1 per cell.

Three observations. First, the reactive and proactive cells post the same headline clean rate (9/10) with radically different underlying health — precisely the capability-vs-integrity distinction of Section 3, now reproduced *between remedies*. Proactive headroom eliminated the agent-layer starvation class outright: zero recorded first-attempt starvation events across the

cell, with the cell's single starvation event occurring in the adversarial layer (a synthesis role call, recovered by that layer's own policy without touching the ten primary agents) — which is why the cell is 8/10 first-attempt clean: nine agent-clean repetitions, one of which recovered that single adversarial-layer event. Reactive headroom left the pipeline sick and survived by paying twice: 96 recorded first-attempt starvation events (6–13 per repetition) — with 95 of 96 events recovered on the second attempt, one repetition lost when the second attempt also starved, and no repetition first-attempt clean. Its observed agent-clean mean was the lowest of any treated configuration (46.9/48), although the small, treatment-conditioned samples are not sufficient to attribute that quality difference to the intervention. Second, the mechanisms are not interchangeable, and the losses show why: the retry cells lost repetitions to double starvation, which proactive headroom prevents, while the bundle's single residual starvation event (repetition 9) was recovered by the retry — without it the bundle would have posted 9/10. That division of labor, prevention of the class plus recovery of its residue, is the argument for shipping the bundle. The proactive cell's loss requires a caveat we state from the implementation: the studied retry fires only on an *empty reply*; a raised transport exception (here, a transient DNS resolution failure) bypasses it and falls through to the deterministic fallback. The transport class is therefore covered by *neither* ablated mechanism — the bundle's 10/10 simply did not encounter one — and absorbing it would require a separate transport-recovery policy, which we note as future hardening rather than claim. Third, the reactive cell exhibited a symptom retry structurally cannot catch: replies truncated mid-JSON, which are non-empty and therefore never trigger the empty-reply check, surfacing instead as parse failures downstream. Starvation has more than one presentation, and only removing its cause — rather than recovering from its most visible symptom — addresses all of them.

	Baseline	Pure retry	Reactive	Proactive	Bundle
Initial budget	baseline	baseline	baseline	headroom	headroom
Recovery on empty	none	retry, same budget	retry +2,000	none[‡]	retry +2,000
Agent-clean repetitions	4/10	9/10	9/10	9/10	**10/10**
First-attempt clean repetitions	—[†]	0/10	0/10	8/10	9/10
Agent-layer starvation events	not instr. (≥6)	102	96	**0**	1
Adversarial-layer starvation events	75	66	0	1	0
Failed debates	3	2	3	2	**0**
Answer-key mean over agent-clean reps	47.8/48	47.6/48	46.9/48	47.8/48	47.2/48
Cause of DEGRADED reps	starvation	double starvation	double starvation	transport (DNS)	—

Table 6: Component ablation, DeepSeek v4-pro direct, n=10 per cell, identical benchmark. "Reactive"/"Proactive" abbreviate reactive-headroom recovery and proactive headroom. Baseline and Bundle columns repeat Table 4. [†]The baseline had no agent-layer retry and predates retry-level instrumentation, so the tier is not classified there. [‡]The agent-layer retry is disabled; the adversarial layer keeps its own production recovery policy in all cells. The reactive and proactive cells matched their registered predictions; the pure-retry cell exceeded its predicted range (9/10 agent-clean versus a registered 3–7/10) and met its registered falsifier threshold — see the discussion of what was actually falsified.

The pure-retry cell fired its own falsifier — and what it falsified is instructive. We registered that a re-sample at the unchanged budget "changes nothing but the dice": 3–7/10 agent-clean, most first-attempt starvations unrecovered, with ≥9/10 as the falsifier threshold.

The cell finished 9/10 agent-clean (boards 48×6, 47×2, 46; one repetition lost to double starvation), with 102 first-attempt starvation events nearly all recovered by an identical same-budget re-request. The registered falsifier fired, and we honor it literally: we reject our registered prediction that a same-budget retry would be insufficient to restore agent-clean availability. What survives is a narrower causal result: proactive headroom was the only tested condition in which recorded agent-layer first-attempt starvation did not *occur*, rather than being recovered from. The observed recovery was far stronger than the exchangeability assumption encoded in the prediction allows; the assumption, not the prevention result, is what the data overturned. Because successful first attempts were not denominator-instrumented, this experiment does not estimate the *magnitude* of the first-versus-second-attempt asymmetry — only its direction: conditional on a recorded first-attempt starvation, the immediate identical re-request rarely starved in this cell. The asymmetry is consistent with provider-side handling of repeated requests (the second request hits the provider's prompt cache; the first does not), but we cannot verify the mechanism from the client side and flag it unresolved. Two practical readings survive. First, the headroom attribution stands on the measurement the ablation was built for: proactive headroom is the only configuration in which the starvation class does not *occur* (0 events), where every retry variant merely recovers from it (96–102 additional retry attempts per cell) with added latency and residual double-starvation losses. Second, no reliability posture should be built on an undocumented, unexplained provider asymmetry — the pure-retry cell "works" for reasons nobody can currently name, which is precisely the kind of silent dependence this paper argues against. A related observation from the same runs: adversarial-layer starvation at the *unchanged* production configuration varied from 0 to 66 events across cells run hours apart (Table 6). The adversarial configuration was unchanged, but its prompts embed upstream agent outputs, so identical request bytes across cells cannot be assumed; the swing is consistent with substantial temporal and/or input-dependent serving variance, and we report it as measured without attributing it.

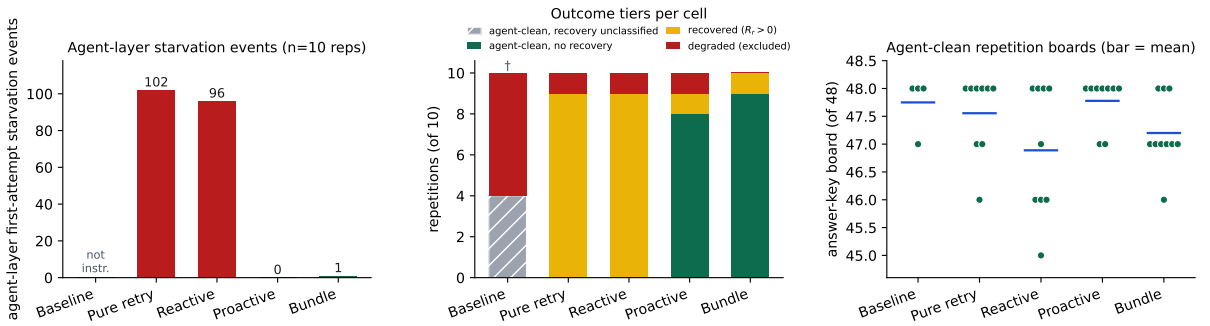

Figure 1: The five configurations at a glance. Left: agent-layer first-attempt starvation event counts per n=10 cell (baseline not instrumented at this layer). Center: repetitions by outcome tier (agent-clean split into no-recovery and recovered; the baseline's four agent-clean repetitions are shown recovery-unclassified because retry-level instrumentation postdates them; degraded excluded from the model's distribution). Right: answer-key scores of agent-clean repetitions, one mark per repetition.

The ablation cells cost $3.01 in total, including two runs lost to infrastructure interruptions, bringing the full study — 58 completed repetitions across two vendors — to $13.35.

8 The Measurement Lesson

The reliability repair did not merely change availability; it changed what could legitimately be *attributed* to the model. Four of six DEGRADED baseline repetitions nevertheless posted perfect 48/48 boards, because deterministic fallback output remained eligible for scoring. After the intervention, all ten repetitions were agent-clean and recurrent model-specific misses became directly observable, enumerated in Table 7. We note explicitly what the means do *not* show: the 47.8/48 mean of the four baseline agent-clean repetitions and the 47.2/48 mean of the ten bundle repetitions should not be read as a causal effect of fallback removal — those four baseline repetitions used no agent fallback by definition, and the samples differ in size and selection. The stronger result is attributional: before degradation accounting, identical or near-identical capability scores could represent materially different execution histories.

Check identity	Protocol	Baseline misses	Bundle misses
`C3-4-threshold` (threshold condition)	AID	0	4
`C3-1-polarity` (savings polarity)	AID	2	2
`C1-grounded` (grounded total)	AID	0	1
`S2.4` (no invented people)	SCI	1	1
`O2.2` (no invented company names)	KPA	1	0
`O3.1` (process-owner attribution)	KPA	1	0

Table 7: Named check misses by run (all repetitions, DeepSeek). In the baseline, misses scatter and partly reflect which agent happened to be starved; in the bundle run, the model's own recurring failure shapes become observable, dominated by one threshold check. Identities match failure taxonomies sketched in earlier characterization rounds of the same pipeline.

We take this as the study's central lesson: in silent-fallback architectures, reliability repair is measurement repair. Until the fallback's contribution is either eliminated or accounted, the system's evaluation numbers describe a model-plus-understudy ensemble whose composition varies invisibly from run to run. Degradation accounting is what makes the composition visible (D_r next to C_r); fixing the degradation is what makes the numbers meaningful.

9 Cross-Model Test: A Third Failure Class

To probe whether the intervention transfers beyond one vendor, we pointed the identical benchmark at a second flagship reasoning model (Kimi K3, direct vendor channel), a model that reasons at maximum effort by default [16]. The first attempt failed comprehensively and honestly (Table 8): 0 to 2 of 10 agents reached the model per repetition, no debates ran, and every completed repetition carried the DEGRADED stamp. Absent the stamps, the understudy-built scoreboards, five of six near-perfect, would have read as the model acing the benchmark.

The cause was neither transport nor token budget but the clock: the model's reasoning outlasted the client's 120-second read timeout on analytical prompts, so calls died client-side while, as far as the billing records show, the vendor continued generating the abandoned replies. This is a third failure class with its own axis (Table 9), and it carries a cost pathology the others lack: the spend is invisible because nothing usable ever arrives.

With a 600-second read-timeout floor for thinking providers, the timeout failure disappeared immediately and both repetitions completed agent-clean (Table 10). We then stopped deliber-

Rep	AID	SCI	KPA	M&A	Agents	Debates	t (s)	Class
1	8/8	11/11	14/14	15/15	0/10	0/0	165	DEGRADED
2	8/8	11/11	14/14	15/15	0/10	0/0	151	DEGRADED
3	8/8	11/11	14/14	15/15	2/10	0/0	208	DEGRADED
4	8/8	11/11	14/14	15/15	0/10	0/0	154	DEGRADED
5	7/8	11/11	14/14	15/15	1/10	0/0	161	DEGRADED
6	8/8	11/11	13/14	15/15	1/10	0/0	141	DEGRADED

Table 8: Kimi K3 pre-fix run (120-second client read timeout), stopped after six repetitions. The model was almost entirely absent; the deterministic understudy produced near-perfect boards; every repetition was correctly rejected by the DEGRADED stamp. Cost of the unusable run: $1.89; billing records (account balance deltas) were consistent with server-side generation continuing after the client abandoned each connection.

Class	Axis	Mechanism	Fix
Transport masking	Infrastructure	Brokered keep-alive padding delivers upstream death as an HTTP 200 (observed in our configuration)	Byte-level detection; one retry; loud on repeat
Budget starvation	Tokens	Hidden reasoning consumes the completion budget; empty reply with length finish	Provider-aware headroom; one retry at the agent chokepoint
Timeout starvation	Wall clock	Reasoning outlasts the read timeout; billing consistent with continued server-side generation	Timeout floor for thinking providers

Table 9: The failure taxonomy. All three classes converge to the same application-level condition (a missing or unparseable reply) and are invisible to status-code checks and scoreboards; each was isolated by a different instrument or controlled comparison.

ately after two repetitions when measured cost ($3.47 per agent-clean repetition) projected a full n=10 beyond the available budget; we report this as a *budget-bounded spot check* (n=2), not a characterization.

Rep	AID	SCI	KPA	M&A	Agents	Debates	t (s)	Class
1	8/8	11/11	13/14	15/15	10/10	3/3	883	agent-clean, recovered
2	7/8	11/11	13/14	15/15	10/10	3/3	927	agent-clean, recovered

Table 10: Kimi K3 post-fix spot check (600-second timeout floor), budget-bounded at n=2, measured cost $6.94. Three starved replies occurred (one in repetition 1, two in repetition 2); all three were recovered by the single-retry policy; zero timeouts, zero salvage. Note the distinct error profile: both repetitions miss a key-person check that DeepSeek never missed in agent-clean repetitions of the bundle run, while repetition 1 clears the AID protocol that DeepSeek missed in 7 of 10 bundle repetitions.

Both repetitions were agent-clean, debate-complete, and parse-clean — and neither was first-attempt clean: repetition 1 recorded one recovered starvation and repetition 2 recorded two

(R_r=1 and R_r=2; the run log attributes all three to the agent layer). Under the taxonomy of Section 3, the spot check is therefore 2/2 agent-clean and 0/2 execution-clean, with every failure recovered rather than absent — on this provider, at these budgets, the retry is load-bearing. This provides preliminary evidence that the instrumentation and remedies transfer across providers; the n=2 spot check is not sufficient for characterization, and we do not claim provider generality beyond it. Two observations worth recording even at n=2: the second model's answer-key misses land on a protocol the first model never missed, while it cleared checks the first model habitually missed (different models, different blind spots, same instruments); and its measured cost per agent-clean repetition was roughly 47× the first model's post-fix figure (Table 11) against a 17× output list-price gap, because always-on maximum reasoning bills every hidden token as output. Reasoning appetite compounds list price; projections built from price sheets alone will err in the expensive direction.

10 Cost

Model (channel)	In $/M	Out $/M	$/agent-clean rep	Basis
DeepSeek v4-pro (direct)	0.435	0.87	0.074	post-fix run: $0.74 / 10 clean reps
Kimi K3 (direct)	3.00	15.00	3.47	post-fix spot check: $6.94 / 2 clean reps
Claude Opus 5	5.00	25.00	—	not measured
Claude Fable 5	10.00	50.00	—	not measured

Table 11: List rates (input at cache-miss) [15, 16, 17] versus measured cost per *agent-clean* repetition of the 48-check benchmark, with the denominator stated explicitly (of those agent-clean repetitions, DeepSeek's bundle run was 9/10 execution-clean; K3's spot check was 0/2 — both repetitions required recovery, so a per-fully-clean figure is undefined there). Under the alternative per-*attempted*-repetition denominator, the two DeepSeek runs give $1.51 / 20 = $0.076. The K3/DeepSeek agent-clean ratio (3.47/0.074 ≈ 47×) far exceeds the 17× output list-price ratio: reasoning appetite compounds list price.

The two DeepSeek runs together consumed $1.51 on the direct channel; the cross-model work added $8.83 including the pre-fix run that exposed the timeout class; the three ablation cells added $3.01 including two attempts lost to infrastructure interruptions. The entire study, 58 completed repetitions across two vendors with full degradation accounting, cost $13.35. Two consequences follow. First, reliability studies at this rigor are within any team's budget; the barrier is practice, not price. Second, Table 11 sharpens the open question our instrumentation makes answerable: what the expensive end of the market actually buys, check by check, once fallback contamination is removed and reasoning appetite is priced in. We leave the full cross-model ladder to future work; the component-wise intervention ablation is reported in Section 7.

11 Limitations

This is an experience report from a single production system, and we state its boundaries plainly. The benchmark uses one fictional corpus with answer keys written by the system's own developers; keys and corpus are frozen, but they are not independent of the system under test; the results should not be read as an independent estimate of model capability — the benchmark

exists to expose attribution failure under controlled repetition in one fallback-enabled pipeline, and this paper is a measurement-method case study, not a model leaderboard. Pre-registration throughout is code-history pre-registration (commits in a private repository, hashes published below), which establishes ordering within our own history but is not independent third-party registration; future rounds should register externally. Sample sizes are n=10 per configuration for the main result and n=2 for the cross-model spot check. The component ablations attribute prevention of first-attempt starvation to proactive headroom at n=10 per cell on one model; the reactive cell's retry administers headroom, so the pure-retry cell is the only budget-free recovery test, and the small, treatment-conditioned samples bound how finely quality differences between cells (46.9 versus 47.8 of 48) should be read. The intervention was validated at n=10 on one model family and corroborated by the spot check on a second; cross-provider claims are preliminary. The transport finding is an observation of one brokered configuration, not a claim about aggregator architectures generally, and the timeout-billing observation is inferred from balance deltas rather than token-level vendor telemetry. The three failure classes we document are specific mechanisms, not a claim of exhaustiveness; other transports and serving modes will have other pathologies (the third class was discovered only when we pointed the instruments at a new vendor, which is itself the point). What we believe generalizes is the method: per-run degradation vectors, fail-closed exclusion of degraded repetitions, controlled channel comparisons, and pre-registered interventions are portable to any pipeline with a fallback path.

12 Conclusion

Three failures that looked identical from the outside had disjoint mechanisms on three different axes, different fixes, and one shared property: all were invisible to status codes and scoreboards. A pipeline that quietly covers for its probabilistic layer will report reliability it does not have and accuracy that is not the model's. The underlying principle is broader than this study: *output correctness does not establish execution integrity.* A system can return 48/48 while the nominal AI component was unavailable, truncated, timed out, salvaged, or substituted — a distinction that matters for evaluation, procurement, service-level agreements, audit trails, and cost attribution alike. The remedy costs almost nothing: record a degradation vector next to every capability score; exclude rather than discount; verify transport at the byte level; register predictions before testing them. On our system this discipline turned a misleading near-perfect scoreboard into an observed 10/10 agent-clean rate (nine of ten first-attempt clean) with an observed, named error profile, for $13.35 of inference. Capability benchmarks tell you which system to choose. Degradation accounting tells you whether to believe what it just did.

Reproducibility. Records for all seven configurations are committed in the project repository alongside the grading harness: six as structured JSON result artifacts (per-repetition boards, event counts, named check failures), and the aborted K3 pre-fix configuration preserved as its original run log. Every empirical results table and Figure 1 is generated from the recorded experiment artifacts. The artifact manifest is:

`audits/10_ai_discovery_pipeline/`, `audits/14_sales_intel/`, … — answer keys (4 protocols)
`audits/4_model_accuracy/bench_four_protocols.py` — grading harness
`results_deepseek_direct_n10_2026-08-08.json` — baseline run
`results_deepseek_direct_n10_headroom_2026-08-08.json` — intervention run
`results_kimi_k3_direct_spotcheck_2026-08-08.json` — cross-model spot check

`results_deepseek_ablation_{retry_only, headroom_only, pure_retry}_part1--5_2026-08-09.json` — the three ablation cells (n=10 each, five run-of-record parts per cell)
`audits/4_model_accuracy/run_logs/` — raw per-repetition run logs, the source of the reported event counts: 20 files comprising the baseline and bundle logs, sixteen ablation logs covering the fifteen run-of-record ablation chunks plus one interrupted, superseded chunk (`abl_A2.log`, replaced by `abl_A2b.log`; the run-of-record reactive cell is `abl_A1, A2b, A3, A4, A5`), and both K3 logs including `kimi_k3_n10.log`, the Table 8 basis (aborted run, log only)
commit `a551e555cedf775e855648e5e5e782c0947e0b1d` — pre-registered prediction (predates the confirmation run)
commit `cee5b11d270f5df76908391a43f9897c954deae1` — spot-check run-of-record
commit `3c56702bce867e937b458473954eb3154811af1e` — ablation levers + pre-registered ablation predictions
commit `4e711f02f53ae099cccc7180150a91aa07ef6fc9` — pure-retry lever + pre-registered prediction

A redacted artifact bundle — the 18 result JSONs, all 20 run logs, and a SHA-256 manifest over every file — accompanies this paper and is publicly available at `https://github.com/VuduVations/mcos-reliability-paper`; the answer keys and corpus are withheld as commercial material, which bounds independent re-grading but not verification of the event records and boards this paper reports. The repository is a private production codebase; beyond the bundle above, remaining artifacts are available to reviewers on request. The corpus is fictional and contains no client or personal data.

Competing interests. The author is the developer and operator of MCOS and holds a commercial interest in VuduVations, the company whose systems are described in this study. No model vendor sponsored, reviewed, or funded this study; all inference was purchased at public list rates from the author's own accounts. Aspects of the pipeline's validation architecture are the subject of pending U.S. provisional patent applications.

AI assistance disclosure. The author used AI-assisted coding and drafting tools during instrumentation, experimentation, analysis, and manuscript preparation. The author directed the work, reviewed the outputs, and is responsible for the paper's claims, data, and conclusions.

References

[1] Y. Xiao, R. Han, Y. Chen, Z. Wang, K. Jiang, Z. CuiZhu, V. Tirumalashetty, W. Wang, B. Gokturk, T. Pfister, C.-Y. Lee. FinanceHarness: Autonomous Financial Deep Research Framework. arXiv:2607.27853, 2026.

[2] P. Liang et al. Holistic Evaluation of Language Models. arXiv:2211.09110, 2022.

[3] X. Liu et al. AgentBench: Evaluating LLMs as Agents. In *ICLR*, 2024.

[4] C. Jimenez et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *ICLR*, 2024.

[5] J. Dean and L. A. Barroso. The Tail at Scale. *Communications of the ACM*, 56(2):74–80, 2013.

[6] G. Candea and A. Fox. Crash-Only Software. In *HotOS IX*, 2003.

[7] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, editors. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, 2016.

[8] D. Sculley et al. Hidden Technical Debt in Machine Learning Systems. In *NeurIPS*, 2015.

[9] W. Kwon et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *SOSP*, 2023.

[10] L. Chen, M. Zaharia, and J. Zou. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. arXiv:2305.05176, 2023.

[11] I. Ong et al. RouteLLM: Learning to Route LLMs with Preference Data. arXiv:2406.18665, 2024.

[12] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. In *ICLR*, 2023.

[13] C. Snell, J. Lee, K. Xu, and A. Kumar. Scaling LLM Test-Time Compute Optimally Can Be More Effective than Scaling Model Parameters. arXiv:2408.03314, 2024.

[14] OpenAI. Reasoning models — API documentation (reasoning tokens are billed as output tokens from the completion budget). `https://platform.openai.com/docs/guides/reasoning`, accessed 2026-08.

[15] DeepSeek. Models & Pricing; Thinking Mode — API documentation. `https://api-docs.deepseek.com`, accessed 2026-08.

[16] Moonshot AI. Kimi K3 — model and pricing documentation (always-on reasoning; `reasoning_effort`). `https://platform.kimi.ai/docs`, accessed 2026-08.

[17] Anthropic. Models overview — pricing (Claude Opus 5; Claude Fable 5). `https://docs.anthropic.com/en/docs/about-claude/models/overview`, accessed 2026-08.

[18] OpenRouter. API documentation. `https://openrouter.ai/docs`, accessed 2026-08.