SAME AGENT, DIFFERENT CONCLUSION: WHY RECOVERY-METHOD COMPARISONS ARE UNDERSPECIFIED

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using agents recover from execution failures by retrying a call or switching tools, which is what lets one finish in a complex environment. Every comparison of such methods is scored by a rule that either counts the calls or ignores them, and neither case is safe. Where they are counted, a repair that should have rescued a failure turns a correct episode into a wrong one: a tool returns empty, the agent retries, and the extra call breaks the match against the gold answer. The effect is not incidental: a reference agent, the middleware it ships with, and a benchmark's checker compose into it, emptying half the tool returns takes it from 0.900 to 0.2625 while emitting 93% more calls; the identical runs scored with recovery calls excluded show no drop. Where they are not counted, the recommended method depends instead on how accuracy is traded against tokens: across $50{,}050$ episodes over nine recovery methods and five benchmarks, four standard ways of making that trade recommend different methods. Both are decidable before an experiment runs: which case applies is stated in the scoring rule, and the weight at which a ranking changes is a function of accuracy and token. A recovery comparison has to state what counts as success, and what success is worth against cost.

1 INTRODUCTION

A tool-using agent acts in an environment it does not control. A return can be empty, an API can time out, an argument can be rejected, and an interface can change after the prompt was written. A reliable agent would not only execute a plan but also recover by retrying a call, switching tools, or replanning what follows. Recovery has become a standard component rather than an accessory as tool-use systems move from controlled benchmarks towards deployment, and a growing set of benchmarks measures it (Yao et al., 2024; Barres et al., 2025; Zhu et al., 2026; Soni, 2026). A natural question follows: does recovery help, and which method recovers best?

That question is not as simple as it looks. Suppose a task requires two tool calls and one tool returns nothing. A sensible agent retries that call and completes the task. If the benchmark scores by comparing emitted calls against gold calls, the retry adds a third call, and the episode is marked wrong although the work was repaired. What failed is the evaluation, not the task.

The inversion points to an explanation simpler than any property of the methods: a conclusion about recovery can be produced by the evaluation. Where a scorer checks final outcome, an extra call costs only compute, so retrying a failed generation is Pareto-efficient on unit-test-scored HumanEval (Kapoor et al., 2025; Chen et al., 2021a). Because calls cannot be retracted, where a scorer compares emitted calls against gold calls (Patil et al., 2025), same extra call changes correctness outright. Which of the two a benchmark is is already written in the scoring rule.

BFCL's AST checker matches emitted calls against gold calls as multisets, so one extra call breaks the match and costs the whole episode. GeoAgentBench's tool-selection score compares the same multisets but grades the mismatch, so extra calls cost in proportion to their number (Yu et al., 2026). Their other rules read the answer, not the trace: BFCL's multi-turn checker scores the object state after execution, GeoAgentBench's parameter accuracy compares gold steps against the last attempt so earlier retries go unseen, τ^2-bench asks whether the gold actions were taken, which extra actions cannot undo (Barres et al., 2025), and ToolFailBench assigns each task an outcome class (Soni, 2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

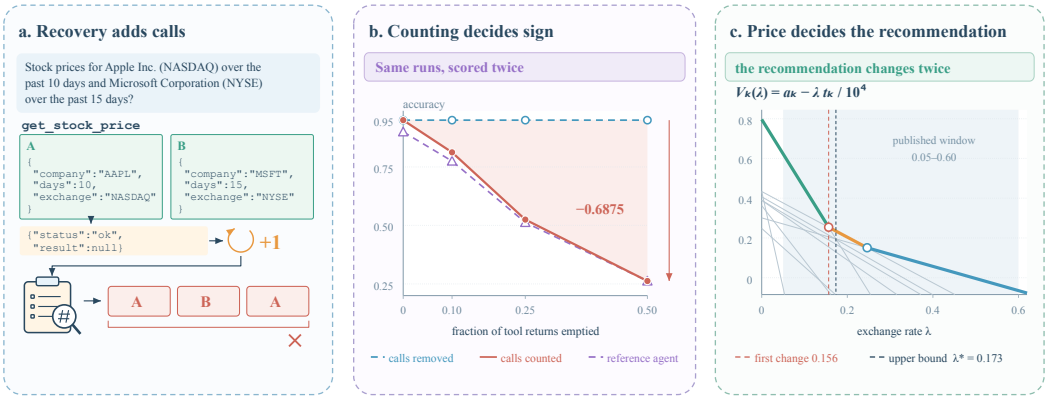

Figure 1: The same recovery looks harmful or beneficial depending on scoring rule. (a) One call return is empty, and retry makes one more. The checker rejects the episode on the count. **(b)** The same episodes scored twice across four injection rates; the recovering policy is specified in Section 3. **(c)** Nine methods on ParallelQA, the highest line at each price being the recommended method. Section 4 gives the counts over all methods.

A controlled intervention separates the two readings. LangChain's reference agent (LangChain, Inc., 2026), its bundled error-handling middleware, and the official BFCL checker each have a defensible design goal alone; composed, they produce the inversion: emptying half the tool returns takes accuracy from 0.900 to 0.2625 while the agent emits 93% more calls. To tell a harder task from a stricter rule, the identical recovery runs are scored twice, once with the recovery calls admitted to the comparison and once without them (Figure 1). The first scoring collapses and the second does not. Tasks, faults, detector decisions and recovery actions are unchanged across the pair, so the same recovery yields opposite conclusions under two evaluations.

Removing that penalty does not say which method recovers best. Once several methods improve outcomes, they separate along another axis, some buying accuracy with more reasoning and retries while others accept slightly lower success at lower cost. The question is incomplete, because comparison needs something it rarely states: what extra accuracy is worth in extra compute?

That is the second place where an evaluation decides the conclusion. For a finished set of experiments, accuracy and token cost can stay exactly as measured while the ranking changes, since it depends on the rate at which one trades for the other. The rate is not only observable by sweeping it: for a given accuracy-cost objective, the point where one method overtakes another follows from the two columns a comparison already prints. Across $50{,}050$ episodes over nine recovery methods and five benchmarks (Yao et al., 2023; Wang et al., 2023; Shinn et al., 2023; Sun et al., 2023), four standard ways of making that trade recommend two or three different methods.

The two phenomena arise at different stages of an evaluation: the first decides whether a recovery is scored as a success, and the second decides which recovery method is worth choosing. The first can reverse the *sign* of recovery's effect; the second can reverse the *order* of the methods. Our contributions are as follows:

- A characterization of when a scoring rule structurally penalizes recovery, obtained by reading the official checkers of current tool-use benchmarks rather than by running them.
- A controlled scoring intervention that isolates this penalty: identical runs are scored with recovery calls counted and removed, with tasks, faults and agent behavior held fixed.
- Across nine methods, five benchmarks and $50{,}050$ episodes, evidence that where recovery is not penalized, the recommended method hinges on the accuracy–cost trade-off; the price at which it changes follows from the accuracy and cost columns already reported.

Whether recovery helps, and which method recovers best, cannot be answered apart from the evaluation. A recovery comparison should therefore state at least two things: **what counts as success**, and **what success is worth against cost**.

2 Problem Formulation

Comparing two recovery methods looks simple: run both and see which scores higher. Yet the conclusion of such a comparison implicitly answers two questions: *what the scorer reads* (Section 2.1)

and *what accuracy is worth in cost* (Section 2.2). Answer either one differently, and the same execution records can support opposite conclusions. Section 2.3 shows when the conclusion changes.

2.1 WHY IS A SUCCESSFUL REPAIR SCORED AS A FAILURE?

An example. A task requires two tool calls, A and B. During execution, A returns an empty result, and the agent retries A and completes the task correctly. The scorer, however, matches the emitted calls against the gold answer: the agent issued A, B, A, whereas the gold answer contains only A, B, so the extra A makes the episode wrong. The task was solved, yet the score is zero (Figure 1(a)).

What the scorer sees. An episode leaves two things behind: the tool calls the agent emitted, and its final answer. Since it is the repeated A that the scorer penalizes, multiplicities must be kept. We therefore record the emitted calls as a multiset C, and the calls in the gold answer as a multiset G.

Which of the two does a rule read? Benchmarks differ: some match C against G, others check only the final answer. We separate the two classes with a binary indicator $\alpha \in \{0, 1\}$, setting $\alpha = 1$ when the score can be affected by calls in C that do not appear in G, and $\alpha = 0$ when it depends on the final answer alone. Since α is a property of scoring rule rather than of how agent runs, it can be determined by reading the scorer's code, which Table 1 does for six rules shipped with current suites.

Measuring the extra calls. Treating a multiset as a map from calls to multiplicities, let $(C \cap G)(x) = \min\{C(x), G(x)\}$ and $(C - G)(x) = \max\{C(x) - G(x), 0\}$. We count matched and extra calls by

$$m(C, G) \coloneqq |C \cap G|, \qquad u(C, G) \coloneqq |C - G|,$$

so that $m = 2$ and $u = 1$ in the example. Holding the final answer fixed and suppressing it from notation, we call a scoring rule s *call-sensitive* if $s(C, G) = h\big(m(C, G), u(C, G)\big)$ for some h that is non-decreasing in m and non-increasing in u. Exact matching is call-sensitive, since $\mathbf{1}[C = G] = \mathbf{1}[m = |G|]\,\mathbf{1}[u = 0]$, and so are graded terms such as precision $m/(m + u)$.

Recovery in symbols. Let C_0 denote the calls the agent would emit without recovering. Recovery (a retry, a tool switch, a replan) appends a collection of calls Δ, so that

$$C = C_0 \uplus \Delta, \qquad \Delta \neq \emptyset,$$

where $\uplus$ denotes multiset sum, under which multiplicities add.

Proposition 1 (Call-sensitive rules)**.** Let s be a call-sensitive scoring rule and $C = C_0 \uplus \Delta$ with $\Delta \neq \emptyset$. Suppose recovery adds no gold call that C_0 lacks, i.e. $C \cap G = C_0 \cap G$. Then $u(C, G) = u(C_0, G) + |\Delta|$ and $s(C, G) \leq s(C_0, G)$. If moreover $C_0 = G$, the condition holds automatically, and under exact matching $s(C, G) = \mathbf{1}[C = G]$ episode turns from correct to wrong.

Let $a_k(\alpha)$ denote the accuracy of method k under a rule of class α. Whenever the rule strictly penalizes extra calls and method k's recovery appends calls on episodes that would otherwise be scored correct,

$$a_k(1) \;<\; a_k(0). \tag{1}$$

2.2 WHEN RECOVERY IS NOT PENALIZED, WHICH METHOD IS BETTER?

Under a rule with $\alpha = 0$, recovery is not penalized directly, yet the comparison does not become simpler: one method may buy higher accuracy with more reasoning and retries, while another is

Table 1: Six scoring rules with current tool-use suites. $\alpha = 1$ where the score depends on emitted calls the gold answer does not contain, and $u(C, G)$ counts those calls (Section 2.1).

Scoring rule	What the score reads	α	Cost of an extra call
BFCL, AST checker (Patil et al., 2025)	call multiset, exact match	1	the whole episode
GeoAgentBench, tool selection (Yu et al., 2026)	call multiset, graded	1	grows with $u(C, G)$
BFCL, multi-turn checker	state after execution	0	none
GeoAgentBench, parameter accuracy	gold steps vs. last attempt	0	none
τ^2-bench, action check (Barres et al., 2025)	whether gold actions are taken	0	none
ToolFailBench (Soni, 2026)	outcome class per task	0	none

slightly less accurate but far cheaper. Accuracy alone cannot decide between them, and prior work accordingly prices cost into the comparison (Wang et al., 2024; Vuddanti & Chittiprolu, 2026).

Pricing cost. Let t_k denote the average number of tokens method k consumes per episode, and let $\lambda \geq 0$ be an exchange rate: the amount of accuracy that 10^4 tokens are worth. The additive form score of method k is its accuracy minus its priced cost,

$$V_k(\lambda) \;\coloneqq\; a_k \;-\; \lambda \,\frac{t_k}{10^4}, \tag{2}$$

where a_k abbreviates $a_k(\alpha)$ once α is fixed. Larger λ means more weight on cost.

A geometric view. Once the results table is fixed, V_k is a line in λ with intercept a_k and slope $-t_k/10^4$: the more tokens a method consumes, the faster its line falls. At a given λ, the highest line is the method most worth choosing at that price, and where two lines cross, the comparison recommends different methods on either side (Figure 1(c)).

2.3 When does the conclusion change?

Both questions determine the conclusion. The method a comparison recommends is

$$k^\star(\alpha, \lambda) \;=\; \arg\max_k \Big[\, a_k(\alpha) \;-\; \lambda \,\frac{t_k}{10^4} \,\Big]. \tag{3}$$

α selects which accuracy column is used, and λ sets how that column is traded against cost. Because α changes a_k and no value of λ can undo that change, the two must be declared separately. Since α is fixed by the scoring rule, remaining question is at what price the recommendation changes.

Locating the change. At $\lambda = 0$ the most accurate method is recommended; as $\lambda \to \infty$, the cheapest. Unless the two coincide, the recommendation must change in between. Several methods may tie for the highest accuracy; let k_{acc} be the cheapest among them, and let k_{cheap} be the cheapest method overall. Setting $V_{k_{\mathrm{acc}}}(\lambda) = V_{k_{\mathrm{cheap}}}(\lambda)$ and solving for λ gives

$$\lambda^\star \;\coloneqq\; \frac{\big(a_{k_{\mathrm{acc}}} - a_{k_{\mathrm{cheap}}}\big) \cdot 10^4}{t_{k_{\mathrm{acc}}} - t_{k_{\mathrm{cheap}}}}. \tag{4}$$

The numerator is the extra accuracy k_{acc} delivers and the denominator the extra cost it incurs, so $\lambda^\star$ is exactly the price at which the former pays for the latter.

Proposition 2 (Reordering bound)**.** Let k_{acc} be the cheapest of the most accurate methods and k_{cheap} the cheapest method overall. If $k_{\mathrm{acc}} = k_{\mathrm{cheap}}$, then k_{acc} maximizes $V_k(\lambda)$ for every $\lambda \geq 0$, and the recommendation does not depend on the price. Otherwise $t_{k_{\mathrm{cheap}}} < t_{k_{\mathrm{acc}}}$, and k_{acc} is not the sole maximizer of $V_k(\lambda)$ for any $\lambda \geq \lambda^\star$.

The bound is one-sided: a third method, intermediate in both accuracy and cost, may overtake k_{acc} before $\lambda^\star$, so $\lambda^\star$ guarantees that recommendation has changed *by* that price, not that it holds *until* then. Proofs of both propositions are in Appendix A. α and $\lambda^\star$ are available before run: the former is written in scoring rule, and the latter follows from the accuracy and cost a paper already reports.

3 Does the Scorer Penalize Recovery?

Proposition 1 says that under call-sensitive rules, recovery can only keep score unchanged or lower it. This is arithmetic, and it leaves three questions open: does the penalty arise with *published* components, does a measured loss *belong* to rule rather than to faults that made task harder, and what is *reading* α worth?

3.1 How can the penalty be made visible?

Tasks and models. We score with BFCL's AST checker on its parallel category, a typical $\alpha = 1$ rule (Patil et al., 2025). Of its 200 tasks, the first 120 were used to develop detector and the repair ladder; we report only on the remaining 80, since how often recovery triggers, and the size of the penalty, depends on development choices. The models are `deepseek-v4-flash` and `kimi-k2.6`.

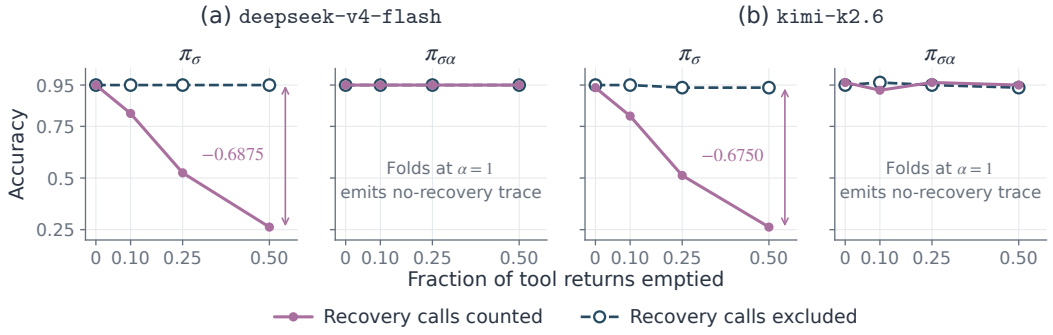

Figure 2: Effect of the scoring rule on recovery. Each panel sweeps the fraction r of emptied tool returns and scores one set of traces twice: *counted* keeps the recovery calls, *removed* drops them before comparison with the gold answer. π_σ follows a fixed repair ladder; $\pi_{\sigma\alpha}$ reads α first and does not repair when $\alpha = 1$.

Injecting faults. We empty a tool's return: the call reports success but carries nothing. Whether a call is emptied is a deterministic function of the task, the tool, the arguments and the occurrence index, so all configurations meet same fault on same call, while a retry counts as a new occurrence and may come back filled. The fraction r of emptied returns is swept over $\{0, 0.10, 0.25, 0.50\}$. Since BFCL does not execute calls and its gold answers assume no faults, this is a *stress test*: it mimics common deployment case in which an interface occasionally returns nothing and retrying is the right response.

Three configurations. A detector flags a return that carries an error key, a failure status, or only empty values; it never consults the model.

- π_σ (*fixed repair*): re-issues the call, then substitutes a tool from the same name family, then replans the remaining steps. Each rung is decided by the return and the tool names alone, so no part of the repair can be tuned toward the result (Figure 3; Appendix C).
- $\pi_{\sigma\alpha}$ (*scorer-aware*): reads α first and does not repair when $\alpha = 1$; otherwise it selects among the same rungs, plus an argument-repair rung, by ascending cost.
- *Control*: the detector records which returns fail, but nothing is repaired.

Neither method is offered as a better recovery: a rule can only be caught penalizing repair if something repairs, and π_σ is the instrument that does. The control tests the alternative explanation, since a loss would belong to faults rather than to scoring. Each set of traces is scored twice, with the recovery calls *counted* and with them *removed*; the only difference between the two is the rule.

3.2 How large is the loss, and whose is it?

It arises with published components. LangChain's reference agent with its built-in error-handling middleware (LangChain, Inc., 2026), run unmodified under the official checker, falls from 0.900 to 0.775, 0.5125 and 0.2625 as r grows, while its emitted calls rise by 93%, from 230 to 444. Each component is reasonable on its own; composed, they turn repaired episodes into failures. Our instrument reproduces the pattern: counted, π_σ falls from 0.950 to 0.8125, 0.525 and 0.2625 (Figure 2), and on `kimi-k2.6` from 0.9375 to 0.2625 (Appendix B).

It belongs to the rule. With the recovery calls removed, the same traces score 0.950 at every rate on both models, and the control also holds at 0.950: the faults did not make the task harder. At each rate the counted score equals the removed score minus the episodes in which recovery triggered on an otherwise correct episode, $76 - 11 = 65$, $76 - 34 = 42$ and $76 - 55 = 21$, with zero residual. Proposition 1 makes this identity necessary, so it validates the instrument rather than revealing a new effect; the removed score stays at 76 of 80, so recovery restores what the fault took and nothing more. At $r = 0.50$, all 59 episodes that triggered recovery are scored wrong, 55 of which would otherwise have been correct, and all 21 that did not are scored correct: under exact matching, any extra call fails the episode. A paired McNemar test gives $p < 10^{-16}$ on both models (Appendix B).

> **The loss belongs to the rule, not to the task.** Identical traces fall from 0.950 to 0.2625 when recovery calls are counted and lose nothing when they are removed, while a recovery-off control is unaffected by the faults.

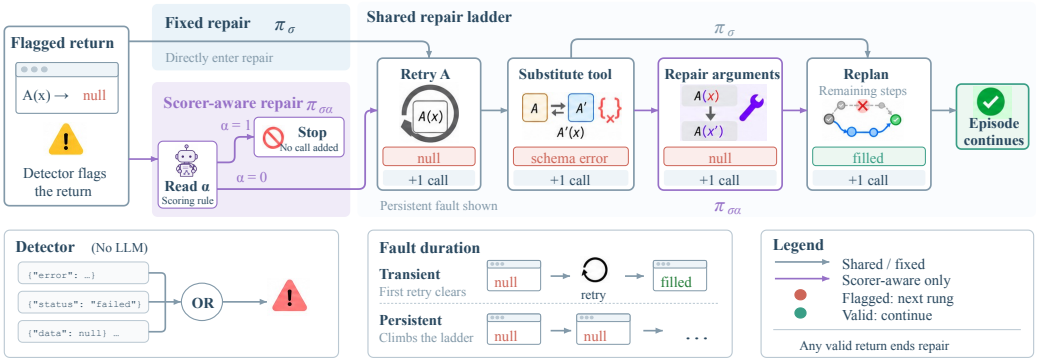

Figure 3: The repair ladder on one flagged return. Each rung issues one more call and shows the return it gets; a persistent fault, drawn here, climbs the ladder, while a transient one clears at the first retry. π_σ enters the ladder directly and skips the argument repair; $\pi_{\sigma\alpha}$ reads α first, stops at $\alpha = 1$, and otherwise takes every rung.

Reading α is enough to decide. $\pi_{\sigma\alpha}$ emits 233 calls and scores 0.950 at every rate, with traces identical to the control's, so its value is measured without either of our repairs on the scale. Its gain is what the rule pays for not repairing, and a comparison that credits it convicts rule rather than method. However, never recovering is not the lesson: on a workload whose scorer reads only the answer, never recovering scores 0.168 against 0.622 for always recovering. The two workloads differ in more than α, so only the sign carries over: the same choice is right under one rule and wrong under the other.

> **The scoring rule, not the quality of recovery, decides whether recovery helps.** Under a call-sensitive scorer the best policy is to read α and decline, and the same choice fails where only the answer is scored. Which method recovers best is a separate question, taken up in Section 4.

4 Does the Recommendation Depend on the Price?

Where scorer reads only answers, an extra call costs tokens only, so which method recovers best depends on what tokens are worth (Wang et al., 2024; Mirzaei, 2026). Proposition 2 says recommendation must change somewhere between free and prohibitive tokens. It leaves two questions: does change fall at prices we actually uses, and if so, why has the published metric not reported it?

4.1 How can a price dependence be observed?

Methods. Nine methods span the ways a repair decision can be made (Appendix C). In three, the model sits in the control flow at every step: ReAct (Yao et al., 2023), SC-k3, self-consistency at k=3 (Wang et al., 2023), and PaR, the plan-and-react hybrid of GeoAgentBench (Yu et al., 2026). In two, it is consulted only once a step has failed: Reflexion (Shinn et al., 2023) and AdaPlanner (Sun et al., 2023). In three, a decidable predicate over the trace triggers the repair: Sherlock's verification placement (Ro et al., 2025) and our π_σ and $\pi_{\sigma\alpha}$ (Section 3.1). KAIJU (Guerin & Guerin, 2026) plans a full call graph once and never repairs, which fixes the floor.

Benchmarks. The benchmarks are Movie Rec. and ParallelQA (Kim et al., 2024), FinQA (Chen et al., 2021b), BFCL-HO, the 80 held-out tasks of Section 3 under the official AST checker (Patil et al., 2025), and ToolMaze (Zhu et al., 2026). BFCL-HO is the only one with $\alpha = 1$; the other four score the answer they define, a recommendation, a number, an entity or a task outcome, and carry $\alpha = 0$. ToolMaze contributes four complexity bands, C1 to C4, each crossed with five injection modes: no fault, a transient failure, a permanent one, a silent empty return, and a return that is well-formed and wrong. One band under one injection mode is a *cell* of 100 tasks, so ToolMaze has 20 cells per model. A *panel* is one benchmark or band under one model, a column of Table 2 or of its counterpart for the second model: eight per model, the four external benchmarks and ToolMaze's four complexity bands, sixteen in all, and a band's panel averages its five cells. Across the 50,050 episodes, every method sees the same tasks, tool backends and fault schedule.

The price window. We sweep λ over $[0.05, 0.60]$, the window the published recovery-efficiency metric declares and sweeps, with its main experiments at 0.5 (Vuddanti & Chittiprolu, 2026); the

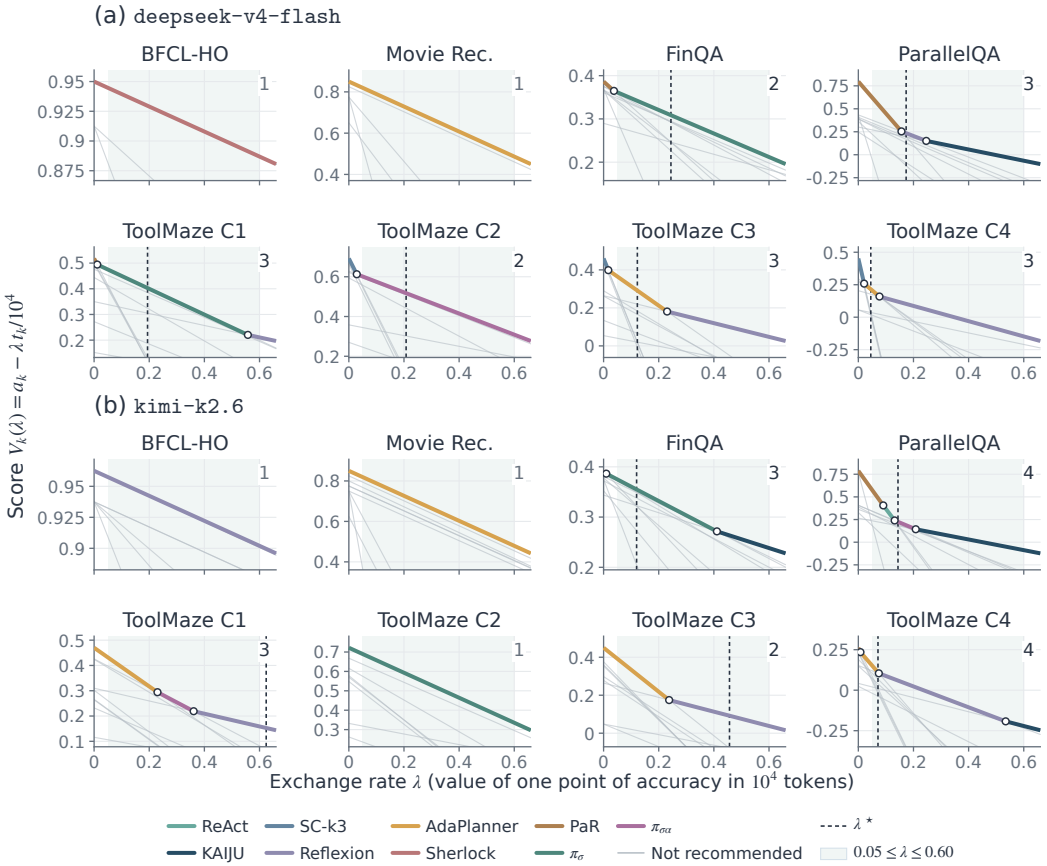

Figure 4: Recommended method as the exchange rate varies, for nine methods under two models. Each panel draws the line $V_k(\lambda)$ of every method and colours the highest one at each price, with the number of distinct recommended methods at its top right; circles mark where the recommendation changes.

interval is theirs, not ours. At $\lambda = 0.05$ a percentage point of accuracy is worth 2000 tokens per episode, and at $\lambda = 0.60$ only 167, a factor of twelve apart. The window therefore spans deployments from one where a wrong answer pays for a long repair to one where the same repair is billed against every request at volume, so a change inside it is a change inside the range the field already looks at.

4.2 Does the recommendation change at prices in use?

On most panels, it does. Before the price reaches the top of the window, the recommended method has changed on 11 of the 16 panels (Figure 4). Resampling the tasks moves these counts little: their 95% intervals are $[9, 12]$ and $[8, 11]$, and both rise when our two methods are removed from the pool (AppendixF), which also repeats test on ToolMaze's 20 cells per model.

The change point can be read off the table. Equation 4 needs two rows of Table 2 and no further episode. On every panel where the recommendation changes, the first change falls at or below $\lambda^\star$, as Proposition 2 requires, and on ParallelQA the bound is close to tight (Table D.2). $\lambda^\star$ is undefined only where the most accurate method is also the cheapest, $k_{\text{acc}} = k_{\text{cheap}}$: on BFCL-HO under either model and on Movie Rec. under `deepseek-v4-flash`, the most accurate methods tie and the cheapest is among them, so no price changes the recommendation.

The recommendation can be chosen, and the choice has a price. Only a method on the upper-left edge of the accuracy–cost plane, where accuracy is high for its cost, can be recommended at any price. In Figure 1(c), three of the nine lines ever reach the top; the other six methods are never recommended however tokens are priced. Formally, $V_k(\lambda)$ is linear in (t_k, a_k) with non-weight on cost, so k is recommended at some $\lambda \ge 0$ exactly when (t_k, a_k) is a vertex of the panel's upper-left convex hull. Most panels have three or more such vertices (Appendix F), so declaring a price

Table 2: Accuracy and tokens per episode for nine methods under `deepseek-v4-flash`. Bold marks the best value. The two columns where $\lambda^\star$ is undefined carry no bold: there the most accurate method is also the cheapest. C1–C4 are ToolMaze's complexity bands. The second model is Table D.1 in Appendix D.

	External benchmarks				ToolMaze bands			
Method	BFCL-HO	Movie Rec.	FinQA	ParallelQA	C1	C2	C3	C4
accuracy ↑								
ReAct	0.912	0.775	0.366	0.248	0.498	0.688	0.456	0.438
SC-k3	0.912	0.775	0.369	0.434	0.502	**0.692**	**0.460**	**0.452**
PaR	0.863	0.650	**0.386**	**0.796**	**0.518**	0.664	0.444	0.422
Reflexion	0.950	0.850	0.361	0.434	0.350	0.358	0.264	0.204
AdaPlanner	0.950	0.850	0.375	0.389	0.444	0.594	0.416	0.296
Sherlock	0.950	0.850	0.366	0.363	0.272	0.270	0.134	0.056
π_σ (Ours)	0.950	0.850	0.375	0.389	0.500	0.622	0.290	0.280
$\pi_{\sigma\alpha}$ (Ours)	0.950	0.825	0.366	0.407	0.476	0.628	0.262	0.272
KAIJU	0.950	0.850	0.290	0.301	0.152	0.168	0.054	0.056
tokens ↓								
ReAct	2100	15610	4924	19312	19517	28193	35685	90253
SC-k3	6158	48275	14777	96944	19155	27576	35946	92034
PaR	2850	17704	5743	34684	20805	28360	36815	90841
Reflexion	1055	6064	2715	11488	2323	2698	3609	5867
AdaPlanner	1054	6058	3385	15021	6177	7323	10207	17874
Sherlock	1052	6070	5742	27753	4369	4830	5521	8124
π_σ (Ours)	1053	6064	2718	12761	5013	5448	8702	15808
$\pi_{\sigma\alpha}$ (Ours)	1054	6072	3002	12331	4687	5321	7855	14935
KAIJU	1053	6072	**1789**	**6100**	**1926**	**2350**	**2794**	**4431**
$\lambda^\star$	—	—	0.244	0.173	0.194	0.208	0.122	0.045

chooses among several recommended methods without rerunning an episode. At each provider's list price, a percentage point of accuracy is worth 578 tokens per episode at ParallelQA's $\lambda^\star$ and 2222 at ToolMaze C4's, which on `deepseek-v4-flash` is \$0.23 and \$0.88 per thousand episodes. Choosing the recommendation therefore amounts to pricing a point of accuracy, a figure a deployment can accept or refuse rather than a parameter to sweep.

The recommendation depends on the price, and the results table says where. Between free tokens and the top of the published window, the recommended method changes on 11 of 16 panels, and $\lambda^\star$, computed from two rows of the table, bounds the first change.

4.3 Why has the published metric not reported it?

Four metrics on one table disagree. Accuracy, accuracy per token, the ratio-normalized form (Vuddanti & Chittiprolu, 2026) and $V_k(0.2)$, all computed from the two columns of Table 2, recommend different methods on most panels (Table D.3). Of the four, only the ratio form is swept across the window, so it is the one that could have reported the change.

Its cost term can barely move a score. The published metric puts cost in a denominator that varies over so narrow a range that cost can hardly change the ranking. It divides accuracy by one plus the price times the method's tokens as a share of a token ceiling. We set the ceiling to the panel's most expensive method, the smallest ceiling that bounds every method and therefore the one under

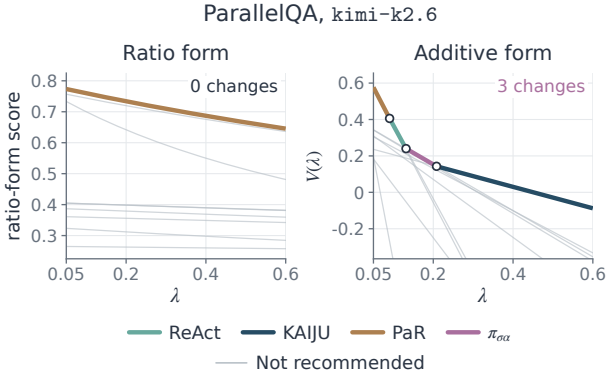

Figure 5: The ratio-normalized metric recommends one method across whole window, while additive form's recommendation changes three times.

which the metric moves most. The share never exceeds one and the published window stops at 0.60, so the divisor stays within $[1, 1.60]$ and cost can shrink a score by at most a factor of 1.60. Two methods can therefore change places only if their accuracies are within a factor of 1.60 of each other; on this grid many pairs sit further apart, fixing their order at every price the metric sweeps. Inside window its recommendation changes on 4 of 16 panels, against 9 under the additive form. Where its recommendation never changes, it is mostly the method already recommended at zero cost (Figure 5).

Its blind spot has a direction. The cheapest method is recommended somewhere in the window on more than twice as many panels under the additive form as under the ratio form (Appendix F). The option a cost-constrained deployment would pick is the one this metric is least able to name, and two deployments that price tokens differently, and are owed different methods, receive the same answer.

> **A ratio-normalized cost term restates accuracy.** Bounded by a factor of 1.60 over its own window, its recommendation changes on 4 of 16 panels where the additive form's changes on 9, and it rarely recommends the cheap method a cost-constrained deployment needs.

5 Related Work

Tool-use benchmarks first scored an agent against a reference: BFCL matches the emitted calls with gold calls (Patil et al., 2025), and τ-bench with its dual-control successor checks the state a conversation leaves (Yao et al., 2024; Barres et al., 2025). Later suites execute the calls, in analysis pipelines (Yu et al., 2026) or under injected tool failures (Zhu et al., 2026), or diagnose how tool use fails (Soni, 2026). They supply the scoring rules this paper reads.

Recovery methods detect a failed step and spend computation on repair: Reflexion turns the error into verbal feedback (Shinn et al., 2023), AdaPlanner revises the plan (Sun et al., 2023), and Sherlock places verifiers where errors are likely (Ro et al., 2025). Repair can cost more than it returns, since intrinsic self-correction can degrade reasoning (Huang et al., 2024) and repeated sampling beats reflection at equal token cost (Mirzaei, 2026). Both results locate the loss inside the method; ours sits outside it, in the rule that scores the trace.

Cost-aware evaluation is established: accuracy alone cannot identify progress (Kapoor et al., 2025), and agents two points apart in accuracy can be nine times apart in cost (Kapoor et al., 2026). Closest to this paper, ERR–ES sweeps an explicit cost weight over $[0.05,\ 0.60]$ in a ratio-normalized efficiency score, reporting an additive score beside it without analysing it (Vuddanti & Chittiprolu, 2026). The weight and the window are theirs. We show that the recommendation already changes inside that window, where the ratio form mostly cannot see it, that a call-counting rule removes score in a way no cost term expresses, and that a bound on where the recommendation first changes follows from two columns already printed.

6 Limitations

First, the measurement at $\alpha = 1$ rests on one benchmark family, because among the five only BFCL scores an episode by comparing emitted calls against gold calls. How far the penalty reaches is therefore settled by reading the other scoring rules rather than running them, which the census does and any benchmark author can repeat for a suite not covered here.

Second, the exchange rate is swept here rather than declared. A sweep shows where a ranking is fragile; a declaration says which method to run and needs a price rule fixed outside the comparison, since a reporter-chosen weight is the degree of freedom this paper studies. Supplying that rule turns the reordering point from a diagnostic into a prescription, and the variable it needs is a per-token price pinned before reading the accuracy and cost columns.

7 Conclusion

Both coordinates of a recovery evaluation can be read before running an episode. Whether a scoring rule charges for emitted calls is stated in the rule itself, and the weight at which a reported ranking turns over follows from the accuracy and cost columns a comparison already prints. A comparison leaving both implicit has not measured the methods alone, and its ranking says as much about the evaluation as about them. What this asks of a recovery comparison is a declaration rather than another metric: what counts as success, and what success is worth against cost.

AI USE STATEMENT

In this work, we used generative AI tools, principally an LLM coding assistant (Claude Code), to implement the methods, the reimplemented baselines and the evaluation harness; to clean and reformat the external benchmarks into a common task format; to formulate the two propositions and assist in writing their proofs. We have not used generative AI tools to generate synthetic data sets, and qualitative or thematic data analysis is not applicable to this work. Additionally, we used generative AI tools to create and edit software code and figures, to search for, identify and summarize related literature, to suggest the structure of the paper, to draft its text, and to format references. We have reviewed all AI-assisted work. Every number the paper reports is computed from the stored episode traces by scripts, and an automated check fails if any of them disagrees with the text; every citation was verified against its primary source; and the authors read and revised the full text. The language models evaluated as agents are the objects of study and are not covered by this statement. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021a. URL https://arxiv.org/abs/2107.03374.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021b. URL https://arxiv.org/abs/2109.00122. arXiv:2109.00122.

Cormac Guerin and Frank Guerin. KAIJU: An executive kernel for intent-gated execution of LLM agents. *arXiv preprint arXiv:2604.02375*, 2026. URL https://arxiv.org/abs/2604.02375.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.01798. arXiv:2310.01798.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Transactions on Machine Learning Research*, 2025. URL https://arxiv.org/abs/2407.01502. arXiv:2407.01502.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S. Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Percy Liang, Arvind Narayanan, et al. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.11977. arXiv:2510.11977.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2312.04511. arXiv:2312.04511.

LangChain, Inc. LangChain. https://github.com/langchain-ai/langchain, 2026. version 1.3.17, langchain-core 1.6.0.

Iliya Mirzaei. Sample more, reflect less: Self-refine and reflexion lose to repeated sampling at equal token cost, from 1.5B to 7B. *arXiv preprint arXiv:2607.28576*, 2026. URL https://arxiv.org/abs/2607.28576.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025.

Yeonju Ro, Haoran Qiu, Íñigo Goiri, Rodrigo Fonseca, Ricardo Bianchini, Aditya Akella, Zhangyang Wang, Mattan Erez, and Esha Choukse. Sherlock: Reliable and efficient agentic workflow execution. *arXiv preprint arXiv:2511.00330*, 2025. URL https://arxiv.org/abs/2511.00330.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366.

Harsh Soni. Toolfailbench: Diagnosing tool-use failures in LLM agents. In *Second Workshop on Agents in the Wild: Safety, Security, and Beyond*, 2026. URL https://arxiv.org/abs/2607.04686.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive planning from feedback with language models. *arXiv preprint arXiv:2305.16653*, 2023. URL https://arxiv.org/abs/2305.16653.

Sri Vatsa Vuddanti and Satwik Kumar Chittiprolu. Recoverability has a law: The ERR measure for tool-augmented agents. *arXiv preprint arXiv:2601.22352*, 2026. URL https://arxiv.org/abs/2601.22352.

Junlin Wang, Siddhartha Jain, Dejiao Zhang, Baishakhi Ray, Varun Kumar, and Ben Athiwaratkun. Reasoning in token economies: Budget-aware evaluation of LLM reasoning strategies. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 19916–19939, Miami, Florida, USA, 2024. URL https://aclanthology.org/2024.emnlp-main.1112/. arXiv:2406.06461.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171. arXiv:2203.11171.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629. arXiv:2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

Bo Yu, Cheng Yang, Dongyang Hou, Chengfu Liu, Jiayao Liu, Chi Wang, Zhiming Zhang, Haifeng Li, and Wentao Yang. GeoAgentBench: A dynamic execution benchmark for tool-augmented agents in spatial analysis. *arXiv preprint arXiv:2604.13888*, 2026. URL https://arxiv.org/abs/2604.13888.

Dongsheng Zhu, Xuchen Ma, Yucheng Shen, Xiang Li, Yukun Zhao, Shuaiqiang Wang, Lingyong Yan, and Dawei Yin. When tools fail: Benchmarking dynamic replanning and anomaly recovery in LLM agents. *arXiv preprint arXiv:2606.05806*, 2026. URL https://arxiv.org/abs/2606.05806.

A PROOFS

Proof of Proposition 1. Step 1: every appended call is already saturated in C_0. Since $C(x) = C_0(x) + \Delta(x)$, the condition $C \cap G = C_0 \cap G$ reads

$$\min\{C_0(x) + \Delta(x),\, G(x)\} \;=\; \min\{C_0(x),\, G(x)\} \qquad \text{for every call } x.$$

Fix x with $\Delta(x) > 0$ and suppose, for contradiction, that $C_0(x) < G(x)$, i.e. $C_0(x) + 1 \le G(x)$. Then

$$\begin{aligned} \min\{C_0(x) + \Delta(x),\, G(x)\} &\ge \min\{C_0(x) + 1,\, G(x)\} && \text{(since } \Delta(x) \ge 1) \\ &= C_0(x) + 1 && \text{(since } C_0(x) + 1 \le G(x)) \\ &> C_0(x) \;=\; \min\{C_0(x),\, G(x)\}, && \text{(since } C_0(x) < G(x)) \end{aligned}$$

which contradicts the condition. Hence $C_0(x) \ge G(x)$ whenever $\Delta(x) > 0$.

Step 2: each appended call is an extra call. For x with $\Delta(x) > 0$, Step 1 gives $C_0(x) - G(x) \ge 0$, so both positive parts below can be dropped:

$$\begin{aligned} (C - G)(x) &= \max\{C_0(x) + \Delta(x) - G(x),\, 0\} \\ &= \big(C_0(x) - G(x)\big) + \Delta(x) \\ &= \max\{C_0(x) - G(x),\, 0\} + \Delta(x) \;=\; (C_0 - G)(x) + \Delta(x). \end{aligned}$$

For x with $\Delta(x) = 0$ we have $C(x) = C_0(x)$, so $(C - G)(x) = (C_0 - G)(x) + \Delta(x)$ holds trivially.

Step 3: counting. Summing the identity of Step 2 over all calls,

$$u(C, G) \;=\; \sum_x (C - G)(x) \;=\; \sum_x (C_0 - G)(x) + \sum_x \Delta(x) \;=\; u(C_0, G) + |\Delta|,$$

while the condition gives directly

$$m(C, G) \;=\; |C \cap G| \;=\; |C_0 \cap G| \;=\; m(C_0, G).$$

Step 4: the score cannot increase. Since h is non-increasing in its second argument and $|\Delta| > 0$,

$$\begin{aligned} s(C, G) &= h\big(m(C, G),\, u(C, G)\big) \\ &= h\big(m(C_0, G),\, u(C_0, G) + |\Delta|\big) && \text{(Step 3)} \\ &\le h\big(m(C_0, G),\, u(C_0, G)\big) \;=\; s(C_0, G). \end{aligned}$$

Step 5: the case $C_0 = G$. The condition holds, since for every x

$$\min\{G(x) + \Delta(x),\, G(x)\} \;=\; G(x) \;=\; \min\{G(x),\, G(x)\}.$$

By Step 3,

$$u(C_0, G) \;=\; u(G, G) \;=\; 0, \qquad u(C, G) \;=\; 0 + |\Delta| \;>\; 0.$$

Because $u(C, G) > 0$, C contains a call in excess of G, so $C \neq G$. Exact matching therefore gives

$$s(C_0, G) \;=\; \mathbf{1}[G = G] \;=\; 1, \qquad s(C, G) \;=\; \mathbf{1}[C = G] \;=\; 0.$$

□

Proof of Proposition 2. Write $a_{\max} \coloneqq \max_k a_k$ and $t_{\min} \coloneqq \min_k t_k$. For any method k,

$$V_{k_{\mathrm{acc}}}(\lambda) - V_k(\lambda) \;=\; \big(a_{k_{\mathrm{acc}}} - a_k\big) \;+\; \lambda\, \frac{t_k - t_{k_{\mathrm{acc}}}}{10^4}.$$

Case $k_{\mathrm{acc}} = k_{\mathrm{cheap}}$. Then $a_{k_{\mathrm{acc}}} = a_{\max} \ge a_k$ and $t_{k_{\mathrm{acc}}} = t_{\min} \le t_k$ for every k, so for $\lambda \ge 0$ both terms are non-negative and k_{acc} maximizes $V_k(\lambda)$ for every $\lambda \ge 0$.

Case $k_{\mathrm{acc}} \neq k_{\mathrm{cheap}}$. Where several methods tie for the lowest cost, k_{cheap} is taken to be the most accurate of them, so $k_{\mathrm{cheap}} = k_{\mathrm{acc}}$ whenever k_{acc} is among the cheapest. Hence k_{acc} is not among them, and $t_{k_{\mathrm{cheap}}} = t_{\min} < t_{k_{\mathrm{acc}}}$. Moreover $a_{k_{\mathrm{cheap}}} < a_{\max} = a_{k_{\mathrm{acc}}}$: otherwise k_{cheap} would be a most accurate method cheaper than k_{acc}, contradicting the definition of k_{acc}. Hence

$$V_{k_{\mathrm{cheap}}}(\lambda) - V_{k_{\mathrm{acc}}}(\lambda) \;=\; -\big(a_{k_{\mathrm{acc}}} - a_{k_{\mathrm{cheap}}}\big) \;+\; \lambda\, \frac{t_{k_{\mathrm{acc}}} - t_{k_{\mathrm{cheap}}}}{10^4},$$

which, since $t_{k_{\mathrm{acc}}} - t_{k_{\mathrm{cheap}}} > 0$, is non-negative if and only if $\lambda \ge \lambda^\star$. For every such λ, k_{cheap} scores at least as high as k_{acc}, so k_{acc} is not the sole maximizer. □

B PER-RATE VALUES OF THE SECTION 3 EXPERIMENT, BOTH MODELS

The appendices call the fault-injection experiment of Section 3 the *probe*. Section 3 reports its two scorings at four injection rates on the first model; Table B.1 gives both models. The *counted* scoring admits the recovery calls to the comparison against the gold answer, and the *removed* scoring drops them from the same traces before the checker sees them, so the two columns of a row are one execution read twice and differ in nothing else. Accuracy is over the 80 held-out tasks, so a cell moves in steps of 0.0125.

Table B.1: Accuracy of the probe on its 80 tasks at each injection rate, with the recovery calls counted and removed. The two halves of a row score one execution twice.

	counted				**removed**			
	r=0	0.10	0.25	0.50	r=0	0.10	0.25	0.50
`deepseek-v4-flash`								
π_σ	0.9500	0.8125	0.5250	0.2625	0.9500	0.9500	0.9500	0.9500
$\pi_{\sigma\alpha}$	0.9500	0.9500	0.9500	0.9500	0.9500	0.9500	0.9500	0.9500
control	0.9500	0.9500	0.9500	0.9500	0.9500	0.9500	0.9500	0.9500
`kimi-k2.6`								
π_σ	0.9375	0.8000	0.5125	0.2625	0.9500	0.9500	0.9500	0.9500
$\pi_{\sigma\alpha}$	0.9625	0.9250	0.9625	0.9500	0.9625	0.9250	0.9625	0.9500
control	0.9625	0.9625	0.9625	0.9375	0.9625	0.9625	0.9625	0.9375

Three readings survive the change of model. The counted scoring takes π_σ from 0.95 to 0.2625 on the first and from 0.9375 to 0.2625 on the second. With the calls removed, π_σ holds at 0.9500 on both, at every rate, so the method solves the same 76 of 80 tasks throughout and the whole movement in the counted column belongs to the rule. And $\pi_{\sigma\alpha}$'s two columns are equal by construction rather than by measurement, because a method that declines to repair leaves no recovery call to remove.

How firm the collapse is. The two scorings read the same traces, so their comparison is paired, and at every rate the discordant episodes fall on one side only: no episode is scored correct with the calls counted and wrong with them removed. There are 11, 34 and 55 discordant episodes on the first model and 12, 35 and 55 on the second at $r = 0.10$, 0.25 and 0.50, which are the identities of Section 3, and a two-sided exact McNemar test on them gives p below 10^{-3}, 10^{-9} and 10^{-16} on both models.

Where the loss comes from. π_σ emits 233, 248, 283 and 334 calls on the first model and 235, 249, 281 and 338 on the second, so the collapse arrives together with a rise of 43% and 44% in emitted calls. $\pi_{\sigma\alpha}$ emits 233 at every rate on the first model, the same 233 the control emits, which is the byte-identical trace Section 3 relies on; on the second it emits 229, 237, 230 and 234 against the control's 231, 230, 230 and 235. LangChain's reference agent in Section 3.2 shows the same pattern with a published implementation: it scores 0.900, 0.775, 0.5125 and 0.2625 while its emitted calls rise by 93%, from 230 to 444.

How much two runs differ. The control and π_σ with recovery suppressed at generation time are two separately executed recovery-off configurations. They agree on 320 of 320 episode outcomes on the first model and 314 of 320 on the second, and the suppressed run reproduces the rescored line, at 0.9500 on every rate for the first model and within one episode of it for the second, so the flat line is not an artifact of how calls are removed from a trace. The same scale of variation places the second model's $\pi_{\sigma\alpha}$ and control rows within two episodes of 0.95, inside the 95% Wilson interval of $[0.878,\ 0.980]$ for 76 of 80.

C METHOD ROSTER AND DETECTOR RATES

Table C.1 lists the nine methods of Section 4, grouped as they are there by what decides a repair.

Table C.1: The nine methods and what each does on a failed step, grouped by what decides the repair.

method	what it does on a failed step
the model is in the control flow at every step	
ReAct	re-plans from the transcript after each observation (Yao et al., 2023)
SC-k3	samples three trajectories and votes (Wang et al., 2023)
PaR	one planning call, then a ReAct-style executor (Yu et al., 2026)
the model is consulted once a step has failed	
Reflexion	turns the error into verbal feedback for the retry (Shinn et al., 2023)
AdaPlanner	revises the plan rather than the step (Sun et al., 2023)
a decidable predicate over the trace decides	
Sherlock	verifies a clean step, re-issues on a failed check (Ro et al., 2025)
π_σ	the fixed ladder of Section 3
$\pi_{\sigma\alpha}$	the same ladder, entered only where $\alpha = 0$
no repair	
KAIJU	its scheduler alone: plans a full call graph once, then executes it (Guerin & Guerin, 2026)

What enters the comparison. Each third-party method enters through the component that decides a repair, which is the object this paper evaluates. PaR is ported from its released code with its prompts intact. KAIJU contributes its plan-execute scheduler, the part of the published system that fixes the call graph before execution, and Sherlock its verification placement; neither component has a runnable public implementation, so both are rebuilt from their papers inside the shared harness. ReAct, self-consistency, Reflexion and AdaPlanner are reimplemented from their papers, and the official ReAct implementation is also run as LangChain's reference agent in Section 3.2.

What every method shares. Within a model, all nine methods run on the same tasks, the same tool backends and the same fault schedule, with the same decoding temperature (0.2 on `deepseek-v4-flash`, 0.6 on `kimi-k2.6`), extended reasoning disabled, and the same output cap per model call (1000 tokens on the probe and the four external benchmarks, 4096 on ToolMaze, where every method also has the same limit of 25 rounds). The token column counts input and output tokens as the provider reports them, summed over every model call an episode makes, so a repair's re-prompt and a reflection pass are both charged to the method that made them. No episode is dropped from any panel, so every method is scored on the full task set.

How often the detector reads a repairable failure. Write f_σ for the fraction of triggers that land on an episode which would have scored correct anyway. It depends on the workload as much as on the detector. On the probe it is 0.932 and 0.915 for the two models at $r = 0.50$, because emptying a return cannot itself change an answer that the checker reads off the emitted calls; on ToolMaze it is 0.074 and 0.053, because there a fault does change the outcome. The probe's high value is why the penalty dominates there: almost every repair is spent on an episode that needed none. Appendix E prices both cases.

D THE SECOND MODEL'S GRID AND THE COUNTS PANEL BY PANEL

Table D.1 is the second model's grid, laid out as Table 2. Tables D.2 and D.3 then give, panel by panel, the quantities behind Sections 4.2 and 4.3, so that each count those sections report can be read off a row. Both are computed exactly from the two columns of the grids, by the same code that draws Figure 4.

Table D.2 carries Section 4.2. $\lambda^\star$ is undefined on three panels and defined on thirteen. On the eleven where it is defined and the recommendation changes, the first change falls at or below $\lambda^\star$ in every case, and on ParallelQA under the first model the bound is nearly tight, 0.156 against 0.173. On the other two, Movie Rec. and ToolMaze C2 under the second model, $\lambda^\star$ is 14.118 and 1.355, so the recommendation holds across the plotted range. Counting the arrows in the last column gives the 14 changes of recommendation inside the window, and the 9 rows with more than one name are the panels whose recommendation changes there.

Table D.1: Accuracy and tokens per episode for nine methods under `kimi-k2.6`. Arrows give the direction of better and bold marks the best value in a column. The column where $\lambda^\star$ is undefined carries no bold: there the most accurate method is also the cheapest, so the recommendation is fixed by a tie rather than by any price. Rows are grouped by what decides the repair, in the order Section 4 introduces them. C1–C4 are ToolMaze's complexity bands, each an unweighted mean over five injection modes at 100 tasks per mode; the external columns carry 80, 40, 352 and 113 tasks. The first model is Table 2.

	External benchmarks				ToolMaze bands			
Method	BFCL-HO	Movie Rec.	FinQA	ParallelQA	C1	C2	C3	C4
accuracy ↑								
ReAct	0.938	0.750	0.378	0.770	0.266	0.548	0.360	0.230
SC-k3	0.938	0.750	0.375	0.770	0.264	0.570	0.354	0.240
PaR	0.938	0.625	0.352	**0.788**	0.302	0.578	0.376	**0.262**
Reflexion	0.963	0.800	0.381	0.363	0.310	0.332	0.264	0.152
AdaPlanner	0.963	**0.850**	0.389	0.389	**0.470**	0.616	**0.450**	0.250
Sherlock	0.963	0.775	**0.392**	0.327	0.232	0.262	0.048	0.024
π_σ (Ours)	0.938	0.775	0.389	0.407	0.424	**0.722**	0.278	0.190
$\pi_{\sigma\alpha}$ (Ours)	0.938	0.825	0.372	0.407	0.426	0.668	0.298	0.150
KAIJU	0.963	0.750	0.344	0.265	0.116	0.172	0.048	0.042
tokens ↓								
ReAct	1894	25461	4742	40235	7847	11017	14372	34764
SC-k3	5634	76832	14217	114922	7883	10979	14229	34448
PaR	2581	26273	6642	42191	8245	11441	14986	35256
Reflexion	1008	6386	2659	11592	2519	2734	3766	6443
AdaPlanner	1031	6180	3327	15921	7646	8173	11587	19628
Sherlock	1009	**6127**	5790	28854	4674	5192	5938	8564
π_σ (Ours)	1014	6162	2863	13075	6509	6445	9559	17044
$\pi_{\sigma\alpha}$ (Ours)	1015	6171	2867	12687	5732	6213	9404	15533
KAIJU	1010	6134	**1757**	**5886**	**1975**	**2387**	**2785**	**4384**
$\lambda^\star$	—	14.118	0.120	0.144	0.624	1.355	0.457	0.071

Table D.3 carries Section 4.3. The four metrics recommend more than one method on 12 of the 16 rows, two or three distinct methods where they disagree. The four rows on which they agree are the two BFCL-HO panels and the two Movie Rec. panels, and on each of them Table D.2 finds no price inside the window that changes the recommendation: the metrics agree only where the exchange rate has nothing to decide.

E THE THREE-FACTOR DECOMPOSITION

Section 2 reads a comparison through two inputs, α and λ, because the paper measures two columns. Splitting the change in reported value from a single repair into three terms shows why nothing else on the evaluation side is needed, and why the penalty dominates the probe:

$$\Delta V_k \;=\; \underbrace{(1-f_\sigma)\,W_k\,v}_{\text{upside}} \;-\; \underbrace{\lambda\,c_k/10^4}_{\text{cost}} \;-\; \underbrace{f_\sigma\,\alpha\,\mathbf{1}[\Delta_k \neq \emptyset]\,m}_{\text{damage}}, \tag{5}$$

$$W_k \;=\; \sum_c p(F_c \mid \sigma, E^-)\,P(\text{cure} \mid k, F_c).$$

Here f_σ is the fraction of triggers that land on an episode which would have scored correct anyway (Appendix C), E^- is the complementary event that it would have scored wrong, v is the value of turning such an episode right, c_k the tokens k's repair adds, Δ_k the calls it adds as in Section 2, and m the score a counting rule removes once Δ_k is non-empty. W_k averages the ladder's cure rate over the fault classes F_c a trigger could be reporting.

Table D.2: Where the recommendation changes, panel by panel. k_{acc} is the cheapest of the most accurate methods and k_{cheap} the cheapest method overall; the first change is the smallest price at which the recommendation changes, and the last column lists the recommended methods across $[0.05,\ 0.60]$ in order. Where $k_{\mathrm{acc}} = k_{\mathrm{cheap}}$, $\lambda^\star$ is undefined. On every panel where it is defined and the recommendation changes, the first change falls at or below $\lambda^\star$, as Proposition 2 requires.

Panel	k_{acc}	k_{cheap}	$\lambda^\star$	first change	recommended across the window
`deepseek-v4-flash`					
BFCL-HO	Sherlock	Sherlock	—	—	Sherlock
Movie Rec.	AdaPlanner	AdaPlanner	—	—	AdaPlanner
FinQA	PaR	KAIJU	0.244	0.038	π_σ
ParallelQA	PaR	KAIJU	0.173	0.156	PaR → Reflexion → KAIJU
ToolMaze C1	PaR	KAIJU	0.194	0.011	π_σ → Reflexion
ToolMaze C2	SC-k3	KAIJU	0.208	0.029	$\pi_{\sigma\alpha}$
ToolMaze C3	SC-k3	KAIJU	0.122	0.017	AdaPlanner → Reflexion
ToolMaze C4	SC-k3	KAIJU	0.045	0.021	AdaPlanner → Reflexion
`kimi-k2.6`					
BFCL-HO	Reflexion	Reflexion	—	—	Reflexion
Movie Rec.	AdaPlanner	Sherlock	14.118	—	AdaPlanner
FinQA	Sherlock	KAIJU	0.120	0.010	π_σ → KAIJU
ParallelQA	PaR	KAIJU	0.144	0.090	PaR → ReAct → $\pi_{\sigma\alpha}$ → KAIJU
ToolMaze C1	AdaPlanner	KAIJU	0.624	0.230	AdaPlanner → $\pi_{\sigma\alpha}$ → Reflexion
ToolMaze C2	π_σ	KAIJU	1.355	—	π_σ
ToolMaze C3	AdaPlanner	KAIJU	0.457	0.238	AdaPlanner → Reflexion
ToolMaze C4	PaR	KAIJU	0.071	0.008	AdaPlanner → Reflexion → KAIJU

Table D.3: The method each of the four metrics of Section 4.3 recommends, panel by panel. The ratio form is evaluated at $\lambda = 0.5$, the published metric's own setting, and every metric breaks ties toward the cheaper method. The four recommend the same method on four of the sixteen rows.

Panel	accuracy	accuracy per token	ratio form	$V_k(0.2)$
`deepseek-v4-flash`				
BFCL-HO	Sherlock	Sherlock	Sherlock	Sherlock
Movie Rec.	AdaPlanner	AdaPlanner	AdaPlanner	AdaPlanner
FinQA	PaR	KAIJU	π_σ	π_σ
ParallelQA	PaR	KAIJU	PaR	Reflexion
ToolMaze C1	PaR	Reflexion	π_σ	π_σ
ToolMaze C2	SC-k3	Reflexion	$\pi_{\sigma\alpha}$	$\pi_{\sigma\alpha}$
ToolMaze C3	SC-k3	Reflexion	AdaPlanner	AdaPlanner
ToolMaze C4	SC-k3	Reflexion	SC-k3	Reflexion
`kimi-k2.6`				
BFCL-HO	Reflexion	Reflexion	Reflexion	Reflexion
Movie Rec.	AdaPlanner	AdaPlanner	AdaPlanner	AdaPlanner
FinQA	Sherlock	KAIJU	π_σ	π_σ
ParallelQA	PaR	KAIJU	PaR	$\pi_{\sigma\alpha}$
ToolMaze C1	AdaPlanner	Reflexion	AdaPlanner	AdaPlanner
ToolMaze C2	π_σ	Reflexion	π_σ	π_σ
ToolMaze C3	AdaPlanner	Reflexion	AdaPlanner	AdaPlanner
ToolMaze C4	PaR	Reflexion	AdaPlanner	Reflexion

The evaluation enters through exactly two symbols, α in the damage term and λ in the cost term; every other quantity belongs to the method or to the workload. That is the sense in which α and λ suffice, and it is why Equation 3 indexes the recommended method by (α, λ) alone.

The first factor is the one an intuition drops. A repair is worth something only on an episode that would otherwise have scored wrong, so the upside is priced on the $1 - f_\sigma$ share of triggers and not on all of them. Ignoring the factor overstates the upside by $1/(1 - f_\sigma)$. On the probe the multiplier is exact rather than approximate, because f_σ is a ratio of episode counts: $11/12$, $34/35$ and $55/59$ at the three rates give 12, 35 and 14.75.

The bit enters multiplicatively, so where $\alpha = 0$ the damage term is identically zero whatever the repair emits, and where $\alpha = 1$ it is charged on the same f_σ share the upside is denied. That is the asymmetry Section 3 measures. On the probe, f_σ near one leaves almost no upside and puts almost all the mass on damage. On ToolMaze the scorer reads the answer alone, so the damage term vanishes, and with f_σ at 0.074 and 0.053 nearly every trigger lands where a repair can win.

The decomposition is an accounting of Equation 3, not a model fitted to the data. Every claim in Sections 3 and 4 is stated on the two measured columns, and v, m and W_k reach those columns only through the accuracy and the tokens each method records, which is why the paper's conclusions need no estimate of them.

F PAIRED-BOOTSTRAP SUPPORT FOR THE COUNTS OF SECTION 4

Section 4 states its findings as counts over panels and cells. Each count is computed from means over 40 to 500 episodes, so this appendix redraws the tasks and asks how far the counts move. Every resample is paired: one set of resampled task indices serves every method in a panel, so two methods are always compared on the same tasks.

The panel counts. Each resample redraws the tasks of all sixteen panels, independently across panels, 1000 times. ToolMaze's five injection modes stay attached to their task, so a band remains a mean over 500 episodes as in Figure 4. Every count of Section 4 is then recomputed on the redrawn columns, with all nine methods and again with only the seven we did not write. Table F.1 gives each count's point value and central 95% interval.

Table F.1: Panel counts of Section 4 under 1000 paired task resamples. Every row counts panels out of 16 except the changes of recommendation, which counts changes.

	all nine methods		**without our two**	
count	point	95% interval	point	95% interval
four metrics disagree	12	[11, 13]	12	[12, 13]
recommendation changes in $[0.05,\ 0.60]$	9	[8, 11]	12	[10, 12]
ratio form changes in $[0.05,\ 0.60]$	4	[2, 7]	6	[3, 7]
additive minus ratio form	5	[2, 7]	6	[4, 9]
cheapest method recommended, additive	7	[5, 8]	7	[6, 8]
cheapest method recommended, ratio form	3	[2, 4]	3	[3, 4]
changes of recommendation in $[0.05,\ 0.60]$	14	[11, 17]	18	[13, 20]
recommendation changes in $[0,\ 0.60]$	11	[9, 12]	12	[11, 13]
three or more methods can be recommended	12	[10, 13]	12	[10, 13]

The comparison Section 4 draws between the two forms holds on the resamples and not only at the point. The additive form's recommendation changes on more panels than the ratio form's in 99.9% of resamples with all nine methods and in every resample with the seven, and the cheapest method is recommended on more panels under it in every resample under both pools. Removing our two methods raises each count of change, from 9 to 12 panels whose recommendation changes inside the window and from 14 to 18 changes in all, so none of these counts rests on the pool containing anything of ours. Proposition 2 needs no interval: it holds for any pair of columns, and therefore on every resample.

The cells. The same test runs one level down, on each of ToolMaze's 20 cells per model, a band crossed with one injection mode over 100 tasks. A cell is counted when $\arg\max_k V_k$ at the two ends of the interval names different methods in at least 95% of the resamples. The test is a lower bound on how often the recommendation changes, since a cell whose recommendation changes inside the interval and changes back before the far end is not counted.

In Table F.2, the second row is again the one to read. Removing our two methods moves the count in opposite directions on the two models, so the cell-level result does not depend on the pool either. The last two rows widen the interval to $[0, 4]$, where the recommendation changes with support in most cells. That is the endpoint behaviour Section 4 sets aside, since a wide enough interval always ends at the cheapest method, and it is the published window that makes the first two rows informative.

Table F.2: ToolMaze cells per model whose recommendation at the two ends of the interval differs in at least 95% of the resamples, out of 20.

cells with support ≥ 0.95	`deepseek-v4-flash`	`kimi-k2.6`
all nine methods	$8/20$	$9/20$
only the seven we did not write	$6/20$	$10/20$
all nine, $\lambda \in [0, 4]$	$15/20$	$17/20$
only the seven, $\lambda \in [0, 4]$	$15/20$	$16/20$

G Reproducibility statement

The two claims the paper turns on can be checked without running any further episode. The value of α is read from a published scoring rule, and Table 1 names the rule and the suite for each value. The reordering bound is a function of the accuracy and token columns a comparison already prints, proved in Appendix A and computable from Table 2 with a calculator.

For the measurements, Appendix C records how each third-party method enters the comparison and the settings every method shares.