SAME TRAJECTORIES, DIFFERENT CREDIT: SCORING-RULE DEPENDENCE IN TOOL-FAULT RECOVERY EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

After a tool call fails and keeps failing, an agent can stop, do the parts of the task the failure did not block, report the failure, or assert the outcome it was asked for. A benchmark's continuation rule decides which of these its score credits, so a higher pass rate does not say which behaviours an intervention suppressed. ToolMaze ships two such checks and evaluates its single-path category c1 with the stricter one; we build matched scorers differing only in the forbidden set and re-score identical stored trajectories. That one change moves the pass rate by 11 to 47 points in every arm, and lowers the benefit credited to a recovery message by 6.6 and 16.0 points on two models. A pre-registered replication on sixty held-out tasks reproduces the level shift but not that reduction: score levels are rule-dependent throughout, whereas the attenuation does not replicate across task sets. Which later sub-tasks count as blocked is not specification-free: three plausible tests label the same 62 relations differently, and the same tool can assert the failed step's outcome in one branch and report the failure in another. Measured apart, the behaviours diverge: the pass rate rises by 33 points while structured-field agreement with a fault-free run on independent sub-tasks falls from 79.4% to 39.7%, and final answers report the failure more often. We recommend reporting the continuation rule with the score, and release the replay protocol, the pre-specified adjudication rubric and the labels.

1 INTRODUCTION

Tool-calling agents are increasingly evaluated on benchmarks that inject a tool fault and score what the agent does next. Any such score has to decide which later actions still count as correct once a call has failed and keeps failing. That decision usually lives in the scoring code rather than in the task. The effect of that choice on the resulting score is rarely measured, because measuring it means holding the agent's behaviour fixed while the rule varies.

In ToolMaze's `C1_task_003`, an agent is asked to build a flight booking request and then to do four further things, none of which consumes any output from it. The booking tool returns `Unavailable For Legal Reasons`, and again on a repeat call. The agent makes the other four calls and reports their results together with the booking failure. The agent's own system prompt tells it to stop when a step cannot be accomplished, so the episode fails, correctly under that policy.

The verdict answers a narrow policy-compliance question rather than distinguishing the post-fault behaviours that produced it. An agent can stop, carry out the sub-tasks the failure did not block, assert the outcome the failed step was asked for, or report that the step failed. The last two are the same tool called with different arguments. All but stopping receive the same failing verdict, because which of them a score credits is fixed by one condition in the scoring code: which calls after the fault-injected tool are forbidden. A higher pass rate therefore does not reveal which kinds of continuation an intervention reduced.

ToolMaze implements two different continuation checks. The check invoked for category c1, whose tasks offer a single tool path, forbids every call to a different tool; a second check in the same commit forbids only the tools that transitively consume the faulty call's output, recording that *independent tools are allowed*, and the c1 evaluation path never invokes it (Appendix K). Which post-fault actions

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.

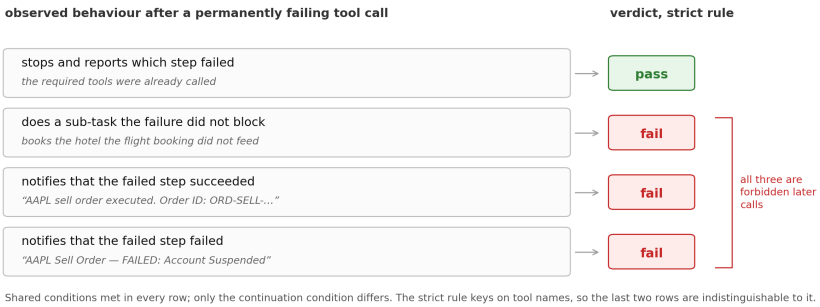


Figure 1: Four behaviours an agent can show after a permanently failing tool call in category c1, and the verdict assigned to each by the continuation rule c1 reaches. Every row is assumed to meet the rule's other conditions, so only the continuation condition separates them. Stopping passes; the other three are later calls to a different tool and fail alike, including the two that differ only in what the notification says. Section 5 measures how often each of the last two occurs and Section 6 what changes between arms.

should remain permissible is a task-specification choice, not a property of the scoring code. Our question is what follows for measurement: Figure 1 shows what the stricter check does to the four behaviours, and we re-score the same logged trajectories under rules differing in that condition alone. The trajectories come from a replay protocol that rebuilds an episode's execution state at the faulty call, so that a runtime message can be varied from a matched start and run to the agent's own final answer (Section 3, which gives the four texts). The credited effect of the message is therefore measured from the state an agent actually reaches when a tool fails, not from the start of an episode.

Swapping the forbidden set moves the pass rate by 11 to 47 points in every arm we measure, with no agent and no trajectory changing. On the forty c1 tasks used here it also lowers the benefit credited to the full message, by 6.6 points on Claude Haiku 4.5 and 16.0 on gpt-4.1-mini (Section 4). We call the two rules *strict* and *data-flow* throughout; the strict rule is the one c1 reaches and the one that produced the recorded verdicts. These are two forms of rule dependence, and we keep them apart throughout: whether the rule changes the absolute scores, and whether it changes the estimated gain of an intervention. Sensitivity of the score levels does not imply equal sensitivity of the intervention effect. A pre-registered replication on the sixty c1 tasks this paper does not otherwise use reproduces the shift in pass rates but not the attenuation of the gain ($+0.9$ $[-1.9, +4.6]$ against $+16.0$ $[8.0, 25.0]$ here): the first form holds in every arm we evaluate, whereas the second varies across the two task sets we evaluate.

Comparing scores under two rules leaves open what a rule ought to forbid. Determining which later sub-tasks count as blocked is not specification-free, and three plausible tests disagree substantially on the same 62 (task, later tool) relations (Section 5). Even fixing one does not settle the matter, because the labels attach to *requested sub-tasks* while the rule sees tool names: the same notification tool asserts the failed step's outcome in some branches and reports the failure in others, a distinction no forbidden set keyed on tool names can draw.

Measuring the behaviours apart shows what the single number leaves out. Between no message and the full one on gpt-4.1-mini the pass rate rises by 33 points, while agreement with a fault-free reference execution on the structured fields of the sub-tasks the failure did not block falls from 79.4% to 39.7%, calls asserting the failed step's outcome become rarer, and hand-coded final answers report the failure more often (Section 6). The pass rate by itself separates none of this.

Contributions. Our main claim is about the reported score: it is conditional on the continuation rule, and byte-identical trajectories receive different credit under rules differing in one condition. A weaker claim concerns the credited *effect* of an intervention: a rule can change that contrast too, but by how much depends on the task set, and our pre-registered replication on the remaining sixty tasks does not reproduce it. "Same trajectory" means one branch rescored under several rules; the message arms themselves generate different trajectories from the same replayed state. We audit the evaluator in three nested layers: the continuation criterion, the dependence relation

it requires, and the behaviours its verdict compresses. (i) How far a benchmark's score, and the benefit it credits to a recovery instruction, depend on the continuation rule, with a pre-registered replication of the intervention-effect result on the sixty tasks the paper does not otherwise use (Section 4, Appendix G). (ii) Whether the dependence relation such a rule needs is uniquely specified: three plausible operationalisations label the same 62 relations differently, and a rule keyed on tool names cannot separate a call that asserts the failed step's outcome from one that reports its failure (Section 5). (iii) What a pass rate still hides once a rule is fixed: a diagnostic panel, released with the replay protocol, the pre-specified adjudication rubric with its sensitivity analysis, and the labels (Sections 6–7).

2 Related work

Construct validity in benchmarks. Prior work has documented construct-validity problems in benchmark evaluation at scale. Jacobs & Wallach (2021) give the measurement-modelling account of the gap between a construct and its operationalisation; Dehghani et al. (2021) show how benchmark and split choices move conclusions; and Bean et al. (2025) review 445 large language model benchmarks against construct-validity criteria and find the gap common. Related concerns are documented in agent evaluation: in evaluation practice (Kapoor et al., 2025), in task-setup and reward-design defects across seventeen agentic benchmarks (Zhu et al., 2025), and in what binary outcomes hide from the logs (Kirgis et al., 2026). Closest to our procedure, Rollout Cards (Masters et al., 2026) re-grade preserved rollout records across benchmarks and report that changing the reporting rule alone moves scores by up to 20.9 points and can invert model rankings.

What this paper adds is a way to isolate the evaluator's contribution to a *measured intervention effect*: the trajectories are held byte-identical while one scoring condition is changed, so the change in credited improvement is estimated directly and with a paired interval, and the relation that condition needs is audited in its own right. ToolMaze provides the controlled setting the comparison needs. This is possible because the benchmark ships both trace checks and the data-flow relation the alternative rule is built on, so the disputed choice is internal to the artifact and is a single condition that can be swapped. To isolate that choice, the scorers of Section 4 retain the shipped check and alter only its forbidden set.

Tool-fault recovery and its evaluation. Tool-agent benchmarks generally score a final state (Yao et al., 2025); ToolMaze (Zhu et al., 2026) defines task success and, separately, a perturbation recovery rate. We audit the trace-based episode criterion behind task success, which is the criterion an intervention is scored on in our runs, and not the separately reported recovery rate; Appendix K gives both and their disagreements. Related benchmarks and detectors (Babu & Iyer, 2026; Babu & Agrawal, 2026; LeVine et al., 2026; Reddy et al., 2026) do report downstream effects, but score the intervention under one fixed criterion. Across the scoring code of seven tool- and web-agent benchmarks, none defines which later sub-tasks a failure blocks and six of seven ship more than one rule (Appendix C): the condition we vary is open across the genre, not local to one artifact. Fault-injection benchmarks of the same generation vary the hazard rather than the criterion (Tian et al., 2026), and one observes that an agent which never calls a needed tool and one which calls it and ignores the result look alike under final-answer accuracy (Soni, 2026). What we vary is the criterion, holding the trajectories fixed.

Runtime messages of the kind we attach have been studied as interventions (Panthi & Abdelfattah, 2026; Nilayam et al., 2026), where the question is which feedback improves recovery rather than what a change in the resulting score identifies; prompt-surface studies show that scores move when the agent's input changes (Sclar et al., 2024; Mizrahi et al., 2024); here the input is fixed and the scorer changes. Karne (2026) argues similarly that a safety monitor's recall has to be read against the behaviour of the model it monitors.

3 Experimental setup and matched-state replay

Environments, faults and agents. The main experiments use tasks `C1_task_001`–`C1_task_040` from ToolMaze (Zhu et al., 2026) category c1, whose tasks offer a single path with no alternative tools. That is the first forty by identifier of the hundred c1 tasks in the release — a

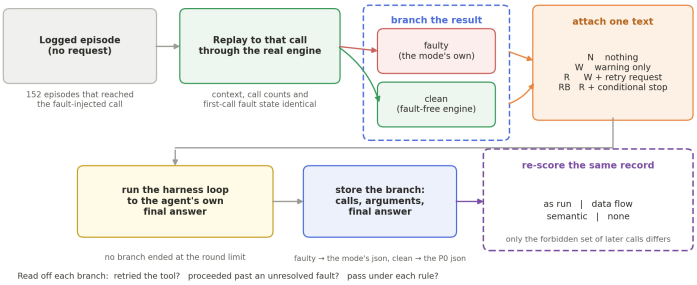


Figure 2: The same-state design and the re-scoring it makes possible. A logged no-request episode is replayed to the fault-injected call; that result is branched into the mode's own faulty result or a clean one, one of four texts is attached, and each branch runs to the agent's own final answer. The stored record is then re-scored under continuation rules that differ only in which later calls are forbidden, so the comparison across rules holds the trajectory fixed.

prefix, not a random sample — capped for cost and not selected on content. Each task runs once per *mode*: P0 with correct tools and four fault modes perturbing one tool's results, explicit or implicit and transient or permanent (P1–P4). Sections 4–6 use the permanent modes (P2 and P4) and the contrast between no runtime message and the full one (N and RB, defined below); the transient modes and the intermediate texts are in Appendix H. The results below use Claude Haiku 4.5 (Anthropic, 2025) and gpt-4.1-mini (OpenAI, 2025) at temperature 0; Appendix I lists every run and Appendix E adds category c2, the same benchmark under a different scoring rule.

In the fault modes ToolMaze gives the agent its fault-aware system prompt unchanged, which already carries the obligation the benchmark scores — stop and report which step failed rather than proceed with incomplete results (Appendix J). Every runtime text we study is therefore an addition to an already fault-aware prompt. Passing requires a repeat call and completion of the required tools on a transient fault and appropriate termination on a permanent one; both are averaged as "success".

Units and statistics. A *task* is one ToolMaze request; a *state* is a task in one fault mode at the point the fault-injected tool was reached; a *branch* is a state continued under one runtime text; a *relation* is a (task, later required tool) pair. Rates are over states unless a denominator says otherwise. Branches from the same task are not independent, so every interval is a 95% *task-resampling* interval: a task-clustered bootstrap (10,000 resamples) with paired comparisons per task. These are descriptive intervals over the evaluated task set, not population confidence intervals for the benchmark, which is a fixed prefix rather than a sample.

Matched-state replay (pre-registered). We take the 152 no-request Haiku episodes in P1–P4 in which the fault-injected tool was reached, rebuild the harness state by re-executing the logged calls up to and including that call through the real execution engine, which reproduces the original context, call counts and perturbation state, and branch that tool's message. The branched result is either the mode's own faulty one or a clean result from a perturbation-free engine, and one of four texts is attached, after which no further text is appended:

N	nothing
W	"AUTOMATED CHECK: this result may be unreliable."
R	W + "Please call this exact tool one more time with the same arguments."
RB	R + "If the second result is usable, continue the task with it; if it shows the same problem, stop and report which step failed."

Each branch runs within the original round budget and is judged against the corresponding task specification; all end with the agent's own final answer, none at the round limit. The 40 tasks yield 152 states that reach the fault-injected call, 76 transient and 76 permanent. Haiku ran all four texts in both regimes (608 faulty branches and 172 clean ones, the clean branches run once per distinct prefix); gpt-4.1-mini ran all four on permanent faults and W and R on transient ones (450 faulty

Table 1: The same permanent-fault branches scored under two continuation rules that differ in one condition: which calls made *after* the fault-injected tool are forbidden. The *strict* rule, the one c1 reaches, forbids every call to a different tool; the *data-flow* rule forbids only the tools identified by the benchmark's own `_compute_dependent_tools` function. Shared conditions are listed below. N attaches no runtime message and RB the full one. The last row is the attenuation A, the paired difference between the two rules' RB−N; positive means the relaxed rule credits the message with less. The third column is the pre-registered replication of Appendix G. Intervals are task-resampling intervals on the paired difference.

Quantity	Haiku 4.5 `001–040` (76 states)	gpt-4.1-mini `001–040` (75 states)	gpt-4.1-mini `041–100` (108 states)
N, strict rule	10.5%	26.7%	36.1%
RB, strict rule	26.3%	60.0%	47.2%
RB−N	+15.8 [6.6, 26.3]	+33.3 [23.7, 43.4]	+11.1 [3.7, 19.4]
N, data-flow rule	57.9%	61.3%	48.1%
RB, data-flow rule	67.1%	78.7%	58.3%
RB−N	+9.2 [0.0, 19.7]	+17.3 [8.1, 27.0]	+10.2 [3.7, 17.6]
Attenuation A	**+6.6** [1.3, 11.8]	**+16.0** [8.0, 25.0]	+0.9 [−1.9, +4.6]

branches). The analysis is conditional on the states the no-request agent reached, and does not estimate a policy applied from the start.

Figure 2 shows the protocol and the re-scoring it makes possible. The four texts generate different continuations from the same replayed state; Appendix H reports what each changes in the agent's behaviour, on all three fault regimes, and at what token cost. The rest of the paper holds each resulting trajectory fixed and varies only how it is scored, so every comparison below is between scoring rules on identical behaviour — the shipped strict check, the benchmark's own data-flow relation, and two further rules — rather than between systems.

4 Scoring-rule dependence of scores and measured gains

For P2 and P4 the benchmark's verdict is exactly its trace check (`_check_p2_legacy`, L303), so the rates in Table 1 are pass rates. The quantity we estimate is

$$A \;=\; (\mathrm{RB}-\mathrm{N})_{\text{strict}} \;-\; (\mathrm{RB}-\mathrm{N})_{\text{data flow}},$$

the *attenuation*: how much of the gain credited to the recovery message is lost when the continuation rule is relaxed, on one fixed set of trajectories. Positive values mean the relaxed rule credits the message with less.

Both rules keep the shipped rule's other conditions: required earlier tools must appear in the call record, only calls after the fault-injected tool's first appearance count, a repeat of it is always allowed, and an abort statement is required when it is the last required step. What the alternative changes is only which later calls are *forbidden*. Neither requires the permitted work to be done, so an episode passes under both by stopping and a higher rate under the relaxed rule is not credit for finishing more of the task. Both columns come from the same stored branches with no exclusions, and the strict column reproduces the harness's recorded verdicts. Because the two rules differ in one condition only, a difference between them is attributable to that condition; the benchmark's own dependency-aware function differs from the strict rule in three ways, and running it unmodified lands close to the data-flow column (Appendix K).

Levels and contrasts move separately. Relaxing the rule reclassifies failed episodes as passes under both texts, and in the first forty tasks more of them under no message than under the full one: on Haiku N rises from 10.5% to 57.9% while RB rises from 26.3% to 67.1%, so the gap between them falls from 15.8 to 9.2 points. The contrast shrinks because the relaxed rule reclassifies more N trajectories than RB ones, not because the message does less. The attenuation needs its own interval rather than the two RB−N intervals read side by side; the last row of Table 1 gives it, task-clustered

and paired. Because the trajectories are fixed across the columns, it measures how much credit an evaluator assigns to the same observed behaviour.

Composition of the attenuation. The reduction in the estimated gain equals the rate at which the relaxed rule reclassifies N episodes minus the rate at which it reclassifies RB ones. It reclassifies 36 of 76 states under N against 31 under RB on Haiku, and 26 of 75 against 14 on gpt-4.1-mini: differences of five states and twelve, which are the 6.6 and 16.0. So relaxing the rule shrinks a measured effect only when it reclassifies the two arms unequally, and nothing in the rule fixes that.

Replication on the remaining c1 tasks. The third column of Table 1 is a pre-registered replication on the other sixty c1 tasks, where the rule reclassifies 13 states under N and 12 under RB: score levels move by 12 and 11 points there as well, and the message still helps, but the reduction is 0.9 points on an interval of $[-1.9, 4.6]$ that contains zero. Score levels change in every arm we evaluate; the reduction in the intervention's estimated gain does not replicate on the additional tasks, so the magnitudes above are as observed on `C1_task_001`–`C1_task_040`. For scale, the rule alone moves levels by 11 to 47 points, against the $+15.8$ and $+33.3$ the shipped rule credits to the message itself.

In category c2, whose criterion credits switching to an alternative tool, the same four texts invert their ordering; c2 differs in topology too, so we read it as consistency evidence only (Appendix E).

The data-flow rule is not self-contained: it needs a relation saying which later actions depend on the failed step. Relaxing the continuation rule therefore does not remove the evaluator's discretion; it moves that discretion into the dependence relation the forbidden set is built from. The next section tests whether that relation is itself uniquely determined.

5 The dependence relation a continuation rule requires

The labels this section constructs support the diagnostic analyses that follow; the rescoring result of Section 4 does not depend on them.

The relaxed rule creates two specification problems. First, it needs a definition of which requested sub-tasks depend on the failed step (Section 5.1). Second, even once that relation is fixed, a forbidden set keyed on tool names cannot separate calls to the same tool that make different claims (Section 5.2).

5.1 Three operationalisations of sub-task dependence

Table 2: Three notions of dependence on the same 62 relations, tabulated against the semantic adjudication as a *reference*, not as ground truth. The last three columns count the relations both call dependent, those only the adjudication calls dependent, and those only the row's notion does.

	labelled		against the adjudication's 29 D / 33 I		
Notion	dependent	independent	by both	semantic only	this notion only
Observed value reuse	4	58	4	25	0
Data flow (benchmark)	21	41	18	11	3
Semantic adjudication	29	33	—	—	—

The rule in Section 4 needs an answer to the first, and three notions of dependence are available, none of them interchangeable. *Observed literal value reuse*: in one fault-free run of the task, a value originating in the failed call reappears verbatim in a later call's arguments. *Data-flow dependence*: the task template's declared `parameter_bindings` route the failed node's output into a later node, transitively. *Semantic dependence*: completing the later sub-task *as the user requested it* requires the failed step to have succeeded — its result is needed, the request made it conditional ("alert me once the booking is confirmed"), or its content asserts the step succeeded ("sell the shares, then email me that they sold"). None of the three measures propagation directly: a notification that falsely reports success need not reuse any value from the failed call.

For the outcome measure of Section 6 we adopt the semantic sense as the operational definition, because the question there is what a user would still have wanted done; that is a choice of perspective, not a claim that the other two are wrong, and Table 9 scores the rule under both. We applied all three to the same rows: every (task, later required tool) pair among the permanent states our branch runs reached, 62 relations over 37 tasks. Independence is a property of the task, the failed tool and the later sub-task, so a task's P2 and P4 share a label. Three of the 40 tasks contribute nothing: in one the agent never reached the fault-injected tool, and in two that tool is the last required step.

Adjudication protocol. Two raters filled the sheet independently under a rubric fixed and released before any label was assigned, each blind to the other's answers and to which runtime text produced any trajectory. **One rater is an author and the other is a language model**, which is how we report it rather than as human evaluation with an inter-rater statistic. They agreed on 53 of 62 rows (85.5%). Seven of the nine disagreements turn on one question the rubric does not settle — whether a sub-task whose purpose the failure defeats still counts as work the user wanted — and we fixed the purposive reading before computing any outcome, reporting the alternative in Appendix B.

Table 2 gives the result. Its 4, 21 and 29 are not progressively more permissive approximations to a single latent ground truth: the semantic and data-flow sets are not nested, 18 relations being dependent under both, 11 under semantic only and 3 under data flow only. The three capture different constructs rather than approximating a single quantity with different errors: observed value reuse sees literal recurrence in one execution, data flow what the template declares, and the adjudication also execution conditions and assertions about the failed step. Against the semantic labels as reference, the value test agrees on 4 of their 29 dependent relations and data flow on 18, with none and 3 false alarms among the 33 independent ones. These are disagreements with a chosen reference, not errors against a validated ground truth.

A rule built on the value test forbids so little that RB−N falls to $+0.0$ $[-5.3, +6.6]$ and $+5.3$ $[0.0, +12.2]$, against $+15.8$ and $+33.3$ under the strict rule; Table 9 therefore carries the other two. The gap between data flow and the semantic adjudication is the one that matters for Section 4, since the data-flow rule uses the template-derived relation: 11 of the 29 relations the semantic adjudication calls dependent are invisible to `parameter_bindings`, most of them cases where the later call's content asserts the failed step's outcome rather than carrying its values.

Structure of the disagreements. Those 14 disagreements (11 and 3 above) are not scattered. Of the 11 the adjudication calls dependent and the template does not, seven are notifications the request asks to send about the step that failed: in `C1_task_037` the user asks for a sell order and then, "after the order is executed", an alert carrying the execution details. No output of the failed order is routed into that alert, so the data-flow rule permits the call while the adjudication marks it dependent. The other four confirm an order never built or a room booked for a meeting never scheduled; the three converse cases are requests that already state the value the failed step would have produced.

Which notion is used changes the measured contrast as well as the labels: scoring the same branches under the semantic relation, and under a rule that forbids nothing, moves RB−N in the order strict, semantic, data flow, none (Appendix B, Table 9).

5.2 Tool identity against call content

The scorer sees concrete calls, whose arguments can change what they claim, and the two questions come apart: a request for a success notification is unsatisfiable once the step has failed, while sending a failure notification through the same tool may be right. A rule that sees tool names applies the sub-task's label to every call of that tool, whatever its arguments, which can change the verdict; we quantify that below.

Call content, and the score's sensitivity to it. "As requested" is the operative phrase, so we read the calls too, under a rubric fixed beforehand. After the same permanently failing sell order in `C1_task_037`, the same `push_alert` sends "AAPL sell order executed …" in one branch and "AAPL Sell Order — FAILED …" in another.

The tool-level label forbids both. Of the post-fault calls to the 29 dependent relations that carry any text, 94 assert that the failed step achieved its requested outcome and 32 report that it failed; all four

text-bearing tools send both kinds, and the rest of the breakdown, including the calls whose schemas carry no message at all, is in Appendix B. What an assertion establishes depends on the fault mode. P2 returns an explicit error, so an assertion is false; P4 returns `success` with a corrupted record, so an assertion is unsupported rather than provably false. The two divide sharply: 31 of 48 decidable P2 calls report the failure, and 77 of 78 decidable P4 calls assert the outcome anyway.

The distinction is not bookkeeping: a rule that excuses a tool whenever *every* one of its post-fault calls reports the failure changes actual verdicts, in the same direction as relaxing the forbidden set. Tool identity is therefore too coarse to encode the distinction; what that establishes is that a tool-level label and a content-level one disagree about which calls to forbid, not that we have a better evaluator to propose. Appendix B quantifies the verdict changes and states the same-author-and-model caveat.

6 Behaviour behind the pass-rate gain

Even a rule that read call content, and so avoided the objection of Section 5, would still return one binary verdict for materially different post-fault behaviours. We therefore measure those behaviours directly between the no-message and full-message arms, and they do not move together.

Pass rate against agreement on independent sub-tasks. Table 3 collects the quantities: the full message raises the pass rate on both models, while over the same branches the sub-tasks the failure did not block match the fault-free reference *less* often. Counting calls is a weak measure of that, since a tool can be called with the wrong arguments, so we re-measured under a stricter bar: called *and* agreeing with a fault-free run on both arguments and result, field by field, with free-text fields excluded because one fault-free run fixes one wording. This is *structured-field agreement with a fault-free reference execution*, not verified task completion, and the exclusion removes exactly the place a false report of success would appear (Appendix A). The share of adjudicated-independent sub-tasks meeting the bar changes by -9.7 $[-22.2, 0.0]$ and -39.7 $[-55.9, -22.1]$ points, against -9.7 and -41.2 from the call count alone, so the fall is not an artifact of counting calls. Only the gpt-4.1-mini interval excludes zero; sub-tasks marked *dependent* fall too under the same bar (Appendix A).

The two reporting channels. In the tool channel both kinds of call become rarer under RB (Table 3), which by itself might mean users heard less about the failure. But the notification is not the only channel. We read all 302 final answers of the N and RB branches under a rubric fixed beforehand and coded each by hand (Appendix F). That channel moves the other way: answers reporting the failure rise by $+17.1$ $[7.9, 26.3]$ points on Haiku and $+18.7$ $[10.7, 26.7]$ on gpt-4.1-mini, and answers asserting the outcome fall by -15.8 $[-25.0, -6.6]$ and -16.0 $[-24.0, -9.2]$; the fall survives a plausible alternative coding of the borderline answers. If a state counts as reported when either audited channel reports the failure, the number of states with no report falls from 33 to 20 on Haiku and from 30 to 16 on gpt-4.1-mini. This is an aggregate improvement in failure reporting, not a guarantee for each state: one state per model loses the report it had under N, and both are identified in Appendix F.

Table 3: What to publish beside the pass rate, from this paper's permanent-fault branches. Row two is structured-field agreement over the states with at least one adjudicated-independent sub-task (36/34); rows three to six read the tool channel and the final answers (Appendix F), coded by one author and, for rows two to four, one language model.

	Haiku 4.5		gpt-4.1-mini	
Reported quantity	N	RB	N	RB
Pass under the strict rule (76/75 states)	10.5%	26.3%	26.7%	60.0%
Structured-field agreement with a fault-free run, independent (36/34)	86.1%	76.4%	79.4%	39.7%
States with a dependent-tool call asserting the outcome	15/76	10/76	11/75	6/75
States with a dependent-tool call reporting the failure	6/76	3/76	2/75	0/75
Final answer reports the failure	56.6%	73.7%	60.0%	78.7%
Final answer asserts the failed step's outcome	42.1%	26.3%	36.0%	20.0%

The diagnostics do not co-move: the pass rate, structured-field agreement, the claims made in the tool channel and the final answer's reporting tell different stories about the same branches.

The same divergence under a different stated policy. The divergence is not peculiar to a runtime message. A pre-registered one-sentence edit of the system prompt, asking the agent to salvage work the failure did not block, raises structured-field agreement by 30.0 points while the strict-rule pass rate falls by 13.8; it raises dependent continuation too, so it is not a clean salvage instance (Appendix D).

7 Discussion and conclusion

7.1 What to report beside a recovery score

Three changes follow for how such a score is reported. **Report the continuation rule as part of the number**: a benchmark shipping more than one should say which is reached and give the effect under an alternative, at the cost of a re-scoring rather than new runs. **Say how dependence was decided**, separately from propagation and from call content, and release the relation itself. **Report the pass-rate change beside structured-field agreement on independent sub-tasks, success assertions and failure reports** (Table 3), with the intervention's cost on fault-free results.

Why score levels move further than measured effects. The two forms of rule dependence come apart for a reason not specific to this benchmark. Relaxing a continuation rule moves a score level whenever it reclassifies many episodes, but moves a *measured intervention effect* only to the extent that it reclassifies the two arms unequally (Section 4). A rule change can therefore move both terms by tens of points and leave the contrast untouched, as the held-out sixty show, so a credited gain has to be measured on the task set at hand.

Where the specification problem recurs. Replacing the stop rule with a dependency-aware one does not remove the evaluator's discretion; it relocates it. The relaxed rule needs a relation saying which later sub-tasks the failure blocks, and three defensible ways of fixing that relation disagree on how many of the same 62 relations are dependent, as do the benchmark's own data-flow relation and the adjudication (Section 5.1). Making the relation sensitive to what a call claims relocates the choice again, into adjudicating claims (Section 5.2), and the same holds a level up, where the choice is which behaviour the rule rewards at all (Section 4). None of this argues for the strict rule or for one relation over another: which later calls remain appropriate is a task-specification question for the benchmark designer to answer (Appendix L).

7.2 Limitations

The evidence is one benchmark family: category c1, permanent faults, two models. The held-out sixty and the c2 check test the reading rather than establish generality, and the attenuation does not replicate on those sixty. The estimand is conditional twice over, on the states a no-request agent reached and on this task set, so it is neither the effect of a policy applied from the start nor an interval over a population. Within c1 the rule changes magnitudes, not which text scores best, so nothing here shows a rule reversing a ranking. Three measures rest on labels rather than execution — the dependence relation and the call-content codes by one author and one language model, the final-answer codes by one author, each under a rubric fixed beforehand with a sensitivity analysis (Appendices B, F) — while the result that carries the paper needs none of them; and structured-field agreement is agreement with one fault-free execution on non-prose fields, not verified completion. Appendix M lists what we withdrew and rerun. Next would come an independent re-adjudication and the same re-scoring on a second benchmark family.

7.3 Conclusion

ToolMaze ships two continuation checks and scores c1 with the stricter one; swapping the forbidden set moves the pass rate by 11 to 47 points and the credited benefit of a recovery message by 6.6 and 16.0. A recovery score measures the rule as well as the agent, and replacing the rule relocates its discretion rather than removing it; the report should carry the rule with the score.

REPRODUCIBILITY STATEMENT

Six stored runs produce the body, all at temperature 0 through their providers' APIs and so deterministic only up to the provider. Appendix I names every run and the file it is stored in, Table 13 maps each reported result to the script that recomputes it, and Appendix H states which runs may not be pooled: the off-prefix replication is its own baseline and branch run, the transient-fault and clean-branch figures come from an earlier Haiku execution whose permanent-fault rates differ slightly from the run the body uses, and the two stated-policy runs are used only for Table 7.

The conversation of every branch in the two permanent runs is retained, which is what makes the re-scoring of Section 4 possible: the four rules are applied to stored traces, not to new executions, and the *strict* column reproduces the harness's own recorded verdicts. `judge.py` is used unmodified for the dependence relation of the data-flow column and for the conditions the four rules share; the rules themselves are in `scripts/rules_by_text.py`. The adjudication sheet, its pre-registered rubric, both raters' labels, the language model's identity and the pre-registered rubric it worked from, and the disagreement resolution are released with the analysis. Code, records, evaluations, pre-registration notes and scripts are in an anonymous repository (`https://anonymous.4open.science/r/same-trajectories-different-credit-80E1`), where claims that a specification was fixed before a run can be checked against the pre-registration notes released with it.

ETHICS STATEMENT

This work re-scores stored agent trajectories under alternative evaluation rules. No new human subjects data is involved and no agent acts outside a sandboxed tool harness. The runtime texts we study raise the rate at which agents repeat a tool call, including on results that were never faulty (Section 3), so we do not present any of them as a recommended intervention. Our reading of ToolMaze's evaluation code describes what the released version does at one commit.

AI USE STATEMENT

AI assistants were used for code, analysis scripts, searches for related work, and drafting and editing the text. A language model was in addition used as the second rater in the dependence and call-content adjudications; its identity, the pre-registered rubric it applied and its labels are released, and the paper reports every result of those adjudications as the work of one author and one model rather than as human evaluation (Section 5, Appendix B). All experimental results were produced by the released code and verified against the raw traces by the authors, who take responsibility for the content of this paper.

REFERENCES

Anthropic. Claude Haiku 4.5. `https://www.anthropic.com/news/claude-haiku-4-5`, 2025.

Rahul Suresh Babu and Adarsh Agrawal. Self-healing agentic orchestrators for reliable tool-augmented large language model systems. *arXiv preprint arXiv:2606.01416*, 2026.

Rahul Suresh Babu and Laxmipriya Ganesh Iyer. Contract2Tool: Learning preconditions and effects for reliable tool-augmented LLM agents. *arXiv preprint arXiv:2606.07904*, 2026.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. In *International Conference on Machine Learning*, 2026.

Andrew M. Bean, Ryan Othniel Kearns, Angelika Romanou, Franziska Sofia Hafner, Harry Mayne, Jan Batzner, Negar Foroutan, Chris Schmitz, Karolina Korgul, Hunar Batra, et al. Measuring what matters: Construct validity in large language model benchmarks. In *Advances in Neural Information Processing Systems 38*, 2025.

Mostafa Dehghani, Yi Tay, Alexey A. Gritsenko, Zhe Zhao, Neil Houlsby, Fernando Diaz, Donald Metzler, and Oriol Vinyals. The benchmark lottery. *arXiv preprint arXiv:2107.07002*, 2021.

Abigail Z. Jacobs and Hanna Wallach. Measurement and fairness. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pp. 375–385, 2021.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Transactions on Machine Learning Research*, 2025.

Sripad Karne. Recall is not protection: Evaluating safety monitors against model compliance. *arXiv preprint arXiv:2609.05797*, 2026.

Peter Kirgis, Sayash Kapoor, Stephan Rabanser, Nitya Nadgir, Cozmin Ududec, Magda Dubois, JJ Allaire, Conrad Stosz, Marius Hobbhahn, Jacob Steinhardt, and Arvind Narayanan. Log analysis is necessary for credible evaluation of AI agents. *arXiv preprint arXiv:2605.08545*, 2026.

Will LeVine, Brendan Evers, Sam Saltwick, and Abhay Venkatesh. RubricRefine: Improving tool-use agent reliability with training-free pre-execution refinement. *arXiv preprint arXiv:2605.09730*, 2026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Charlie Masters, Ziyuan Liu, and Stefano V. Albrecht. Rollout cards: A reproducibility standard for agent research. *arXiv preprint arXiv:2605.12131*, 2026.

Moran Mizrahi, Guy Kaplan, Dan Malkin, Rotem Dror, Dafna Shahaf, and Gabriel Stanovsky. State of what art? a call for multi-prompt LLM evaluation. *Transactions of the Association for Computational Linguistics*, 12:933–949, 2024.

Prashanti Nilayam, Kiran Ramanna, and Prashil Tumbade. Detection without correction: A two-parameter decomposition of multi-stage LLM pipelines. *arXiv preprint arXiv:2605.27559*, 2026.

OpenAI. Introducing GPT-4.1 in the API. `https://openai.com/index/gpt-4-1/`, 2025.

Sugam Panthi and Rabab Abdelfattah. Outcome monitors: Recovery affordances for silent tool failures. *arXiv preprint arXiv:2608.19303*, 2026.

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 *International Conference on Machine Learning*, 2025.

Vikas Reddy, Sumanth Reddy Challaram, and Abhishek Basu. Reason less, verify more: Deterministic gates recover a silent policy-violation failure mode in tool-using LLM agents. In *KDD 2026 Workshop on Evaluation and Trustworthiness of Agentic AI*, 2026.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations*, 2024.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design. In *International Conference on Learning Representations*, 2024.

Harsh Soni. ToolFailBench: Diagnosing tool-use failures in LLM agents. In *ICML 2026 Workshop on Failure Modes of Agentic AI*, 2026.

Yang Tian, Zhengpeng Shi, Yu Zhou, and Bo Zhao. Beyond function calling: Benchmarking tool-using agents under tool-environment unreliability. *arXiv preprint arXiv:2606.25819*, 2026.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

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.

Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, et al. Establishing best practices in building rigorous agentic benchmarks. In *Advances in Neural Information Processing Systems 38*, 2025.

A Failure taxonomy, fault discriminability and value use

Two episodes sharing one verdict. The verdict does not distinguish these outcomes, and both occur. Under no instruction, 68 of Haiku's 76 permanent states fail the as-run rule; in 24 of those (35%; gpt-4.1-mini 22 of 55, 40%) every later sub-task the adjudication marks independent was called with matching arguments and results, and the state had no dependent sub-task at all. In `C1_task_001` the permanently failing flight booking is followed by a Hilton Tokyo lookup and a room booking, both correct and neither taking any input from the booking. At the other end, `C1_task_033` under P4 on gpt-4.1-mini fails with the same verdict for a different reason: the perturbed contact lookup returns the identifier `u_bob` in place of the one requested, and the agent queries that person's calendar — the faulty value reaching a later call, in all four texts. One label, two outcomes a user would not confuse.

The structured-field bar. A sub-task meets it when the tool was called and every non-free-text argument and output field equals the fault-free run's, as strings, after normalising whitespace, numeric format and case. Each state contributes the fraction of its own such sub-tasks that meet the bar, and the reported rate is the unweighted mean over states with at least one.

Independent against dependent sub-tasks under the structured-field bar. The contrast Section 6 reports for adjudicated-independent sub-tasks is -9.7pp $[-22.2, 0.0]$ on Haiku and -39.7pp $[-55.9, -22.1]$ on gpt-4.1-mini; for adjudicated-dependent ones it is -17.9pp $[-33.3, -4.8]$ and -29.1pp $[-40.9, -17.4]$. The difference between the two contrasts is $+8.1$ $[-9.7, +26.5]$ and -10.6 $[-30.3, +8.3]$, so neither model resolves which class the instruction suppresses more (`scripts/structured_field_by_class.py`). In 24 of the 68 Haiku states that fail the as-run rule under no instruction, every adjudicated-independent sub-task met the bar and no dependent sub-task existed at all; the gpt-4.1-mini figure is 22 of 55.

Faults carrying an agent-visible cue. An earlier version of this analysis compared the perturbed output with the clean output of the same tool and counted any difference as a cue. That overcounts: the agent never sees the clean output, so "a populated field became empty" is not by itself something it could have caught, since an empty list may be a legitimately empty result. The classification reported here keeps only cues checkable from what the agent received, in this order (`scripts/fault_discriminability_v2.py`): *error signature*, the result carries an error field or an error-like status; *error token*, an error-like token inside a value, such as an `order_id` of `UNKNOWN-ERR-628`; *empty result*, the whole output empty or all-null; *argument mismatch*, a field echoing an argument the agent passed with a different value, such as a `flight_number` sent as `CA1234` and returned as `CA4321`; *range violation*, a quantity outside its admissible range, such as a negative `character_count`; *context contradiction*, a value contradicting one an earlier tool returned, counted only where that tool was actually called in the same episode and its value received; and otherwise *needs the clean output*. Over the 76 implicit faults the no-request run reached: error signature 3, error token 11, empty result 0, argument mismatch 38, range violation 6, context contradiction 7, needs the clean output 11. A cue is therefore present in 65 of 76 (86%). On the 76

permanent faults the same classification gives 70 with a cue; among those 70, proceeding past the unresolved fault still occurs in 87.1% of the no-text branches, 80.0% under R and 70.0% under RB. The classification bounds what was available to the agent; it does not show that any agent used it.

Table 5 gives the full taxonomy with every category. Its rows are the whole-episode runs, which attach a verification request to every tool result (V0, with V1 and V2 alternative wordings of the same request and V0CMP/V0STOP its closing clause varied) or only to results a detector flags (*gated*), against a no-request arm. Those runs are not re-scored anywhere in this paper; they appear here and in the value-use counts below because they are the widest set of arms on which the taxonomy can be tabulated.

Cost on clean results. On P0 the always-verify request goes on every result: the agent retries in 15 of 40 episodes and completion is unchanged, 33 of 40 against 33 with no request.

Composition of continuation, and scope for value reuse. The benchmark fails a permanent-fault episode for calling any other tool after the fault-injected one, without asking whether that tool needed the failed value. Two questions follow, and they have different kinds of answer.

The first is about one fault-free execution of the task rather than about the agent: *does that run pass the failed call's output to anything later?* Running each task without faults, we take the fault-injected call's output, keep the leaf values that originate there — absent from the user's request and from every earlier call — and ask whether any later call's arguments contain one. Only 6 of the 76 Haiku permanent states and 6 of the 75 gpt-4.1-mini states have such a consumer. In the rest, this test detects no direct reuse of those values in later call arguments — an observation about the single fault-free execution it reads dependencies from, not a property of the task, and silent on the dependencies Section 5 shows it misses.

The second is behavioural: *among the states where something could be carried, was it?* Here the numbers are small enough that we report them as counts, not rates. Under each text, Haiku continues in 2 of its 6 such states and propagates in none of them; gpt-4.1-mini continues in 1 to 3 and propagates in exactly one, the same episode throughout, in which a contact lookup for one person returns another person's user id and the agent queries that person's calendar and messages them. We draw no rate from this. What the two questions together establish is about the score rather than the agents. Of the branches that continued past the fault, 96–97% on Haiku and 94–96% on gpt-4.1-mini are in states where this test finds no later call consuming the failed value in the fault-free run; in the remainder it does, and the value was carried in one episode of gpt-4.1-mini and none of Haiku's. The test reads one fault-free execution, so a consumer it does not find may still exist. The count of continuations is therefore not a measure of propagation.

Counts by text, as stopped / continued / of which reused the value (`scripts/continuation_taxonomy.py`). Haiku, out of 76 states: no text 10/66/0, warning 12/64/0, retry request 17/59/0, conditional stop 24/52/0. gpt-4.1-mini, out of 75: 25/50/1, 24/51/1, 30/45/1, 51/24/1, and 39/29/1 out of 68 for the specific-cue arm. Continuations come from 28 to 34 distinct tasks on Haiku and 21 to 36 on gpt-4.1-mini, so no single task drives them.

Which run each number comes from. The Haiku counts here are computed from the regenerated run (`exp15_perm_regen`), because that is the run whose conversation traces survive, and they are internally consistent: its own continuation counts are the 66, 64, 59 and 52 above. The *transient* and *clean* Haiku figures quoted in Section 3, and all of Table 10, come from the recorded run (`exp15_branch_run2`), whose continuation counts are 67, 64, 61 and 54; the permanent-fault contrasts in Section 3 come from the regenerated run, like the rest of the body. The two runs are the same design executed twice and agree on 95.7% of continuations, but they are not the same episodes, so a pass from one table and a propagation verdict from the other must not be read as facts about a single episode. The gpt-4.1-mini counts and contrasts both come from `exp17_branch`, one run, and can be read together.

An earlier version of this analysis matched JSON field names as well as values, so a later call carrying its own `reservation_id` argument looked like reuse, and reported seven propagating episodes on gpt-4.1-mini where there is one. Whether the final answer tells the user that a step failed is a further question that a keyword search over free text answers badly — "did not confirm

successfully" reports a failure using none of the words such a list holds — so the script reports that column and we draw no conclusion from it; the question is answered instead by the hand-coded audit of Appendix F.

The Haiku permanent traces used here were regenerated by replaying the branches a second time, after a file-naming error in our own tooling overwrote the originals. The metrics were unaffected, being stored separately, and the regenerated branches agree with the recorded run on 97.4% of repeat calls, 95.7% of continuations and 96.4% of verdicts, inside the run-to-run variation reported above.

Continuation, restricted to cue-carrying faults. Of the 76 permanent faults the no-request Haiku agent reached, 70 carry a cue in what the agent received. Restricted to those 70, the agent still proceeds past the unresolved fault in 87.1% of episodes with no text, 80.0% under R and 70.0% under RB — within a point or two of the unrestricted rates. The aggregate pattern is therefore unchanged when the sample is restricted to faults carrying a cue, which is weaker than a statement about why any individual episode stopped.

Value carried after a repeat call. On transient faults the repeat returns the clean result, so the branches can be read for which value the agent then used; Table 4 gives the counts.

Table 4: Repeating the call, passing the benchmark, and which value is used, for the same transient branches. The two halves have different denominators: retry and pass are over all 76 transient branches per arm (75 for gpt-4.1-mini), value use only over implicit-transient branches whose repeat call returned the clean result on a separating field, 27 to 31 per arm as the rows sum. The reference arm differs by model because the second-model run omitted the no-text arm here. "Both" means values from the two results appear in different fields; every such episode was read by hand (Appendix A).

		all transient branches		value carried forward			
Model	Text	repeated	passed	clean	perturbed	both	not echoed
Haiku 4.5	N	27.6%	22.4%	—	—	—	—
	R	94.7%	81.6%	22	0	2	3
	RB	100.0%	88.2%	26	0	3	2
gpt-4.1-mini	W	22.7%	20.0%	—	—	—	—
	R	94.7%	85.3%	21	0	1	5

Value carried forward, by arm. For the implicit-transient episodes in which the agent retried the fault-injected tool and received the clean result, we compare which of the two values appears afterwards, in the arguments of later tool calls and in the final answer (`scripts/value_use_after_retry.py`). A field counts only if it separates the two results: excluded are fields where one value contains the other, and fields the perturbation merely permutes, since in both cases a downstream hit would not say which result was carried. An episode is scored *clean* if a clean value appears downstream and no perturbed value does, *stale* in the reverse case, *both* if values from the two results appear in different fields, and *neither* if the changed fields are not echoed at all. Counts (clean / stale / both / neither / undecidable) are: Haiku always-verify V0 $21/0/3/2/6$ of 32; gated $8/0/1/1/1$ of 11; V1 $5/0/0/0/3$ of 8; V2 $2/0/2/0/1$ of 5; gpt-4.1-mini V0 $22/0/2/6/7$ of 37. In the same-state branches the pattern repeats: Haiku $22/0/2/3$ of 27 under R and $26/0/3/2$ of 31 under RB, gpt-4.1-mini $21/0/1/5$ of 27 under R.

No episode was classified *stale*, that is, carrying the perturbed value and not the clean one; one *both* episode does retain an incorrect confirmation code, described next. The 14 *both* episodes arise from four distinct tasks, and we read all of them: three of the four are collisions rather than staleness — a status field set to `completed`, which the final answer also uses as ordinary prose; a `product_id` that an earlier stock-check call had legitimately returned; and a pair of values, `Meeting` and `Notes`, that are both words of the document title the task creates, which leaves that episode undecidable. One is genuine, in task `C1_task_001`: under V2 a final answer reports the perturbed confirmation code `Z9Y8X7` beside an otherwise clean booking record, so a perturbed field can survive into an episode the trace check credits. An earlier version of this analysis reported the

collisions as staleness; the scripts now separate them. V1 and V2 contribute few episodes because they retry rarely.

Failure taxonomy for every arm. Mutually exclusive categories per episode, every arm (`scripts/failure_taxonomy.py`). Transient = P1+P3, permanent = P2+P4 (80 each); clean = P0 (40).

Table 5: Where recovery fails, all whole-episode arms. Columns: not reached / no retry, fail / retry, pass / retry, fail (transient); not reached / no retry, proceeds / no retry, stops (pass+fail) / retry, proceeds / retry, stops (pass+fail) (permanent); completes / fails (clean).

	Transient				Permanent					Clean	
Arm	n.r.	no retry	retry, pass	retry, fail	n.r.	no retry, proc.	no retry, stop	retry, proc.	retry, stop	pass	fail
Haiku none	4	50	21	5	4	65	8	2	1	33	7
Haiku gated	4	27	44	5	4	29	5	36	6	35	5
Haiku V0	2	10	61	7	2	14	0	50	14	33	7
Haiku V1	3	38	34	5	4	49	3	18	6	34	6
Haiku V2	4	36	34	6	4	56	4	9	7	34	6
Haiku V0cmp	2	14	56	8	2	20	0	45	13	33	7
Haiku V0stop	2	18	53	7	2	28	0	35	15	33	7
gpt-4.1-mini V0	4	1	65	10	5	5	8	45	17	34	6
gpt-4.1-mini V2	4	28	44	4	4	29	17	20	10	35	5

Two trajectories. Task `C1_task_003` under V0, where the third call (`build_flight_booking_request`) is the perturbed one. The two modes inject different first errors and the same request; what they share is the agent's response to it. The modes differ in whether the repeat call clears the fault.

	P1 (transient)	P2 (permanent)
Call 3	{`"error": "Maximum Concurrent Connections Reached"`}, with the V0 request appended	{`"error": "Unavailable For Legal Reasons"`}, with the V0 request appended
Call 4	retries `build_flight_booking_request`; the booking is returned	retries the same tool; the same error is returned
After	weather, balance, exchange rate, notification — **pass**	weather, balance, exchange rate, notification — **fail**: other tools were called after a fault that was never resolved, in a task with no alternative path

Under P1 the retry resolved the fault; under P2 it could not, and what fails the episode is not the retry, which the judge permits, but the downstream calls after the unresolved fault. In the same run the agent also retried on 71% of clean results, each retry a paid tool call and a longer transcript, one component of V0's 1.9× agent-token cost (retries, output length and the subsequent path change together; we do not decompose them).

B Adjudicating which later sub-tasks depend on the failed step

The value-reuse test of Appendix A detects *direct value reuse*. Whether a later sub-task depends on the failed step is a semantic question that test cannot reach, so we had it judged directly, with the protocol fixed and released before any label was assigned (`notes/adjudication_A_prespec`).

Unit and sheet. One row per (task, later required tool) among the permanent states our branch runs reached: 62 rows over 37 tasks. Independence is a property of the task and the failed tool, so a task's P2 and P4 share one label. Three of the 40 tasks contribute nothing: in one the no-request agent never reached the fault-injected tool, and in two that tool is the last required step, leaving no later sub-task to judge. Each row carries the user's request, the failed tool and the later tool, and nothing else. In particular the sheet does not show what the automatic test concluded: the quantity being measured is where judgement and that test diverge, and printing it would contaminate exactly that. Those verdicts were kept in a separate file for the comparison afterwards.

Rubric. **D** if the later tool needs the failed call's result, whether by value, because the request made it conditional on that step succeeding, or because its content asserts that step's outcome as fact; **I** if it could be completed correctly regardless; **?** if the request does not say. Sequencing words ("then", "after that") are not by themselves conditions, and their absence is not by itself independence. A dependence on another later tool propagates only when that tool is itself D. Where the request states the value the failed step would have produced, the later tool that uses it is I.

Raters and resolution. Two raters filled the sheet independently, neither seeing the other's answers nor which runtime text produced any trajectory; one is an author and one is a language model, and we report it as that rather than as human evaluation with an inter-rater statistic. They agreed on 53 of 62 rows (85.5%), with no "?". The nine disagreements were not scattered: seven turn on whether a later sub-task whose purpose the failed step defeats — a room booked for a meeting that was never scheduled — still counts as work the user wanted. The rubric did not settle that class: it asks whether the sub-task can be completed *correctly*, which one rater read as executability and the other as serving the stated purpose. We do not claim the rubric decided it. We fixed the purposive reading for the main analysis, before computing any outcome, and report the alternative for both quantities the labels feed.

On the *semantic column of Table 9*, counting all seven as independent raises the pass rates, as a more permissive rule must, and moves the contrasts by 1.3 points on Haiku and 5.3 on gpt-4.1-mini without changing their direction or their intervals' exclusion of zero: RB−N is $+10.5$pp $[+1.3, +21.1]$ on Haiku and $+18.7$pp $[+10.5, +27.8]$ on gpt-4.1-mini, against $+11.8$ and $+24.0$ under the labels we use, and RB−R is $+6.6$pp $[+1.3, +14.5]$ and $+13.3$pp $[+5.3, +22.7]$ against $+7.9$ and $+16.0$, a move of 1.3 and 2.7. The reduction against the strict rule, which is what Section 4 reports, is if anything larger under the alternative labels: 5.3 and 14.6 points rather than 3.9 and 9.3.

On the *call-count measure*, the same substitution gives RB−N $= -8.3$pp $[-20.2, 0.0]$ on Haiku and -37.5pp $[-52.5, -22.5]$ on gpt-4.1-mini, against -9.7 and -41.2 under the labels we use. On the *structured-field measure* of Section 6, which is what Table 3 reports, it gives -8.3pp $[-20.2, 0.0]$ over 42 Haiku states and -35.0pp $[-51.2, -20.0]$ over 40 gpt-4.1-mini states, against -9.7 $[-22.2, 0.0]$ over 36 and -39.7 $[-55.9, -22.1]$ over 34. The larger independent set lowers the levels slightly (Haiku's no-message rate 86.1% to 85.7%, gpt-4.1-mini's 79.4% to 76.2%) and shrinks both contrasts, in neither case enough to change the reading. What the alternative does not preserve is one detail: under the main labels Haiku's RB−R is numerically identical under the as-run and semantic rules, and under the alternative it is not ($+7.9$ against $+6.6$). Directions, interval judgements and the reduction against the strict rule are unchanged; that coincidence is not. The other two disagreements were rows whose siblings had been filled differently and were resolved by applying the rubric consistently. Six resolutions matched one rater and three the other. The final labels are 29 D and 33 I.

The unit a label attaches to. The unit is a (task, later required tool) pair, so a label forbids or permits *the call*, not a particular content. That is coarser than the distinction it sometimes needs to draw: after a failed order, a notification saying the order was filled and one saying it failed are the same tool. We label the sub-task as the user requested it — an alert reporting the order's execution — so a call whose content instead reports the failure is scored against a label written for the requested version. Tools that bundle functions, such as `snapshot_workspace_and_alert`, inherit the same coarseness: the backup may be independent while the alert is not, and one label covers both. A finer unit would be the call with its arguments; we did not adjudicate the labels at that unit.

What the forbidden calls said. The rubric was fixed before any argument was read (`notes/notification_content_prespec`): **A** the arguments assert the failed step happened, by completion wording or by presenting its execution detail as fact; **B** they report that it failed; **C** they say nothing either way; **D** the call has no natural-language field. Borderline cases go to C or D and never to A, so A is the conservative side. The cue that decided each verdict is stored with it, and the cues are listed per task in `scripts/notification_content_audit.py` so that a reader can see which substring did the work.

The population is every call to one of the 29 adjudicated-dependent tools occurring after the fault-injected tool's first appearance, in both main runs and all four texts: 266 calls, of which **A** 94, **B** 32, **C** 13, **D** 127. Between the two arms of Section 6 the calls move with the states: assertions fall

from 16 to 11 on Haiku and 12 to 7 on gpt-4.1-mini, failure reports from 7 to 4 and 3 to 0. The 127 are `book_meeting_room`, `confirm_order`, `confirm_held_recommendation_order`, `snapshot_workspace_and_alert`, `schedule_meeting`, `book_best_meeting_room`, `query_availability_outlook_calendar`, `update_cart_item`, `submit_order_with_best_discount` and `place_equity_order_from_risk`, whose schemas carry no message: there the call performs the action and there is no text to be right or wrong. Four tools carry text, and all four send both kinds: `push_alert` 24 assertions to 11 failure reports, `send_instant_message` 25 to 9 (with 12 making no claim), `create_document` 20 to 7, and `push_alert_contact_by_name` 25 to 5. In `C1_task_037` the same `push_alert` sends "AAPL sell order executed. Order ID: ORD-SELL-2024-03-10" in one branch and "AAPL Sell Order — FAILED: Account Suspended" in another, after the same permanently failing order.

Restricted to the 11 relations the template does not call dependent — the subset Section 5 rests on — the 107 calls divide **A** 44, **B** 11, **C** 0, **D** 52. We pre-registered that the share of A would fall under the full message and that B would be non-zero in every condition. Neither prediction was supported. On that subset the share of A among decidable calls does not fall (72.7% to 80.0% on Haiku from N to RB, 85.7% to 100% on gpt-4.1-mini) — what falls is the number of such calls, 8 to 4 and 6 to 2 — and B is 0 under R on Haiku and under RB on gpt-4.1-mini. The classification is by the same author-and-model pair as the labels, and it is a judgement about what a message claims, not about whether sending it was right; every argument string is released with its verdict and cue (`results/notification_content_audit.json`).

Incremental effect of the stop clause. Isolating the clause from the retry request it is bundled with, RB−R is $+29.3$ under the strict rule on gpt-4.1-mini and $+16.0$ under the semantic rule, a reduction of 13.3pp $[5.5, 21.6]$; on Haiku the two coincide at $+7.9$ (Appendix B gives why, and why the coincidence is label-dependent). The whole message is rule-sensitive on both models and the clause alone on one of them — a difference between two sets of reached states, not a ranking of the models.

Why Haiku's RB−R coincides across two rules. The as-run and semantic rules disagree outright on 24 to 27 of the 76 Haiku states depending on the text, but the *change* from R to RB is identical on every one of them under both rules, so the paired difference is zero state by state and every bootstrap resample returns 0.0. Under the alternative labels above the coincidence breaks ($+7.9$ against $+6.6$), so it is a property of this sample and this labelling, not of the model.

The content-aware column. Section 5 recomputes the semantic rule with a tool excused when *every* one of its post-fault calls is **B**; a tool with one assertion and one failure report stays forbidden, so the excusal is per tool and per state, not per call. Pass rates rise to 51.3% and 60.5% under N and RB on Haiku and 56.0% and 77.3% on gpt-4.1-mini, against 46.1/57.9 and 53.3/77.3 under the tool-level semantic rule; RB−N falls to $+9.2$ $[-1.3, +21.1]$ and $+21.3$ $[+12.0, +31.1]$ and RB−R to $+7.9$ and $+12.0$ $[+4.0, +21.3]$. Read as attenuations from the semantic column, the content-aware rule gives $+2.6$ $[0.0, 6.6]$ and $+2.7$ $[0.0, 6.7]$ on RB−N, and 0.0 $[-3.9, +3.9]$ and $+4.0$ $[0.0, 9.2]$ on RB−R, so none of the four excludes zero. Verdicts change on 11 of Haiku's 304 state-texts and 8 of gpt-4.1-mini's 300, so the excusals are not absorbed by other forbidden calls in every episode (`scripts/content_aware_rescoring.py`).

Denominators. The 62 relations become state-level counts through the 76 permanent branch states: 36 states have at least one adjudicated-independent later tool, 42 have at least one dependent one, 6 have both and 4 have neither. The two sets overlap: $36 + 42 - 6 = 72$ states have at least one, and the remaining 4 have neither, the fault-injected tool being the last required step, so $72 + 4 = 76$. Sheet, labels, pre-registration and the comparison script are released.

C How related benchmarks score a tool failure

Section 4 varies one scoring condition inside a single benchmark. This appendix asks what the surrounding literature does with that condition. We read the scoring implementation of seven tool-agent and web-agent benchmarks (Yao et al., 2025; Patil et al., 2025; Zhou et al., 2024; Liu et al., 2024; Ruan et al., 2024; Lu et al., 2025; Barres et al., 2026) and coded four things: what the scorer compares, whether a call issued after a tool failure is permitted, what the failure itself costs, and whether the artifact ships more than one scoring rule. Claims come from the scoring code and the official documentation at the revision we read on 16 September 2026, not from the papers, which mostly do not state the behaviour; where a repository has diverged we name the branch. Table 6 omits the second question because the answer is the same everywhere: in all seven, a call issued after a tool failure is executed and scored like any other.

Table 6: Scoring of tool failure in seven related benchmarks, read from their scoring code. "Blocked-step relation" is whether the artifact defines which later sub-tasks a failure prevents, the relation Section 5 has to construct. "Rules shipped" counts scoring rules distributed in the same artifact, not metrics derived from one rule.

Benchmark	What is scored	What a failure costs	Blocked-step relation	Rules shipped
τ-bench	Final database hash; response substrings	Nothing: failures return strings and leave the hash unchanged	None	One
BFCL	Backend state; result multiset	Nothing beyond the step budget	None; a turn-level assumption skips later turns	Two applied, and a strict trace checker present but commented out
WebArena	Last action's answer, URL, page state, multiplied	Nothing: the error text is stored in `info` and never read	None	One harness, combined per task
AgentBench	Varies by environment: state, string, F_1, judge, blends	Nothing for environment errors; on `v0.2` an agent-side format error ends the episode at zero, which `main` reverses	None; the knowledge-graph variable list is an execution mechanism	Eight environments, eight rules
ToolEmu	Language-model judge over the trajectory text	Less than nothing: the judge is told failed risky calls are "certainly not" risky	None	Two judges, never combined
ToolSandbox	Milestone graph against database snapshots	Nothing: the exception column's default similarity is the constant 1	Ordering, with no blocking	One, with a nullifier and a turn count
τ^2-bench	Final database hash times selected side channels	Nothing until ten cumulative errors, then zero	None; golden actions are an unordered set	Eight evaluation types

Three things follow. First, none of the seven defines which later sub-tasks a failure blocks. The relation Section 5 constructs is not one the literature answers differently; it is one it has not asked. Second, six of the seven distribute more than one scoring rule, so the plurality we exploit in ToolMaze is ordinary rather than a peculiarity of that artifact. Third, what a failed call costs ranges from nothing to the whole episode, and that cost is usually a by-product of the implementation rather than a stated decision: WebArena stores the error text and never reads it, ToolSandbox gives the exception its own column and scores it with a constant, and in τ-bench a failed call leaves the scored hash untouched. BFCL is the exception, shipping a strict trace checker, disabling it, and saying why — trace equivalence "often marks recovery behaviors as wrong, unfairly penalizing models for exploration and correct recovery actions."

This is a map of rules, not a comparison of systems, and it does not show any benchmark adopting a rule opposite to ours. What it shows is that the condition we vary is unsettled across the genre and mostly invisible in it.

D THE STATED-POLICY EXPERIMENT

Table 7: The same tasks under two stated system policies — the manipulation is one sentence of the agent's system prompt, not a user message — gpt-4.1-mini, no runtime message. Pass rates are the three rules of Section 4; the last two rows apply the structured-field bar of Section 6 to the sub-tasks the adjudication marks independent and dependent, over the states having at least one of that kind (32 and 44 of 80). All differences are SALVAGE minus ATOMIC; task-paired, task-clustered intervals. "Agreement" is agreement with a fault-free run on non-prose fields, not verified completion of the user's purpose.

	ATOMIC	SALVAGE	SALVAGE − ATOMIC [95%]
Pass, strict rule	33.8%	20.0%	$\mathbf{-13.8}$ $[-22.5, -5.0]$
Pass, data-flow rule	65.0%	56.2%	-8.8 $[-17.5, -1.3]$
Pass, semantic rule	57.5%	48.8%	-8.7 $[-17.5, 0.0]$
Structured-field agreement, independent	60.9%	90.9%	$\mathbf{+30.0}$ $[+15.6, +45.0]$
Structured-field agreement, dependent	38.6%	54.5%	$+15.9$ $[+4.2, +28.1]$

Result and state-level decomposition.

Continuation and score under the two policies. The agent responds to the wording: under SALVAGE it meets the structured-field bar on 30.0 points more of the adjudicated-independent sub-tasks and 15.9 points more of the dependent ones (Table 7). The adaptation runs in the direction asked for but does not avoid increasing dependent activity, and all three rules score that arm lower.

Sources of the score difference. Thirteen states pass under ATOMIC and fail under SALVAGE and two go the other way, the 11 of 80 behind the 13.8 points. Reading each of the thirteen for the condition that actually failed: twelve fail on a forbidden call, one because a required tool never appears in the record, and of the twelve, *seven* have a forbidden call the adjudication marks independent while *five* have only dependent ones. Re-scoring the thirteen is a separate observation, not a consequence of that split, since the two label sets are not nested: seven pass under data flow and five under the semantic rule. Part of the deficit is work the policy asked for and the rule forbids; part is calls on sub-tasks the semantic labels mark dependent, which is a wider class than the wording's own prohibition, so neither that the calls were forbidden by the policy nor that they were inappropriate follows from the label (Section 5).

The manipulation. One sentence of ToolMaze's fault-aware system prompt is replaced; the task-completion and final-answer clauses, and the judge, are untouched. The shipped sentence reads "If you determine that a required step cannot be accomplished by any available tool or approach, stop the workflow and clearly report which step failed and why, rather than proceeding with incomplete results." In its place, ATOMIC reads "The user's request is a single transaction. If a required step cannot be accomplished by any available tool or approach, stop the workflow and clearly report which step failed and why. Do not perform any of the remaining sub-tasks." and SALVAGE reads "The user's sub-tasks are separable. If a required step cannot be accomplished by any available tool or approach, report which step failed and why, and continue with the sub-tasks that do not depend on that step. Do not perform sub-tasks that need the failed step's result." The substitution is made through an environment variable; with it unset the prompt is byte-identical to the shipped one, which we check by string comparison.

Runs. gpt-4.1-mini, the same 40 c1 tasks in P2 and P4, 80 whole episodes per policy, no runtime message. No episode ended in a provider error, at the round limit, or without a final answer.

Which states enter which rate. Unlike the branch experiment, a whole-episode run can end without the fault-injected tool ever being called: that happens in 6 states under ATOMIC and 5 under

SALVAGE, on three tasks. The trace check fails such an episode for a missing required tool, so they enter the pass rates of Table 7 as failures. Restricted to the 74 states where both arms reach the fault, the three contrasts are -14.9 $[-24.3, -5.4]$, -9.5 $[-17.6, -1.4]$ and -10.8 $[-20.3, -1.4]$, slightly larger than over all 80 and with the semantic interval no longer touching zero. We report the unrestricted rates in the body because they are the rates the benchmark would produce, and give the restricted ones here.

The outcome rows use a different denominator again. They require the fault to have been reached, so that later sub-tasks can be read off relative to it, and they require the state to have at least one sub-task of the kind in question: 32 states for independent and 44 for dependent. That is fewer than the 36 of 76 in the branch experiment because those 6 unreached states remove two tasks that do carry independent labels, not because the labels differ — they are the same 62 relations of Section 5, and the task set is unchanged. Each qualifying state contributes the fraction of its own sub-tasks meeting the bar, and only calls after the fault-injected tool count.

Pre-registered predictions and outcomes. Five were fixed before the runs. The pass rate under the strict rule would favour ATOMIC (held, 13.8pp); SALVAGE would complete more adjudicated-independent work (held, 30.0pp); the two together would show the rule scoring the policy that does more of the requested work lower (held as stated, but see Section 6 for why that is weaker than it sounds); the semantic rule would narrow the pass gap (held, $13.8 \rightarrow 8.7$); and SALVAGE would *not* raise dependent completion, since its closing sentence tells the agent not to perform sub-tasks that need the failed step's result. The last failed: dependent completion rose 15.9pp $[+4.2, +28.1]$. Part of the explanation may be that the prediction read the semantic labels as coextensive with that sentence, and they are not: the labels also cover sub-tasks made conditional on the step or asserting its outcome, which the wording does not name. We report it because it is what limits the reading of the rest.

State-level decomposition. Of the 80 paired states, 10 show SALVAGE completing more independent sub-tasks with no increase in dependent ones while the strict rule still fails it, and the semantic rule passes 6 of those 10; 8 show more dependent completion without more independent; 1 shows both; 61 show neither. Scripts and pre-registration are released with the analysis.

E THE C2 REPLICATION: A DIFFERENT SCORING RULE

ToolMaze's category c2 gives each task two valid paths and two to four interchangeable tools, and its judge scores permanent faults accordingly: `_check_c2_permanent` passes an episode when the agent switches to a different alternative tool and the shared tools all succeed. There is no rule against calling other tools after the fault. The continuation analysis of this paper is therefore specific to c1, and we say so rather than inferring it from these runs; it follows from the scoring code.

The replication uses gpt-4.1-mini on all 49 c2 tasks in modes P0, P2 and P4, 146 baseline episodes (one P4 task is absent from the released set). Clean-task success is 73.5%, above the 50% floor fixed in the pre-registration for treating a model as competent on the task set; no episode recorded a provider error as a final answer. Of the 97 permanent episodes, 87 reached the fault, and those states were branched under the same four texts, giving 348 branches with no replay mismatch, no provider error and none at the round limit. The texts are byte-identical to those used on c1, since changing them would break the comparison. The pre-registered endpoints were RB$-$N and R$-$N on benchmark pass, and RB$-$N on switching; the two primary contrasts resolved as predicted and with intervals excluding zero, while R$-$N was a null detection (-3.4pp, $[-10.5, +3.5]$) as its prediction allowed. By mode, benchmark pass runs $45/59/52/32\%$ on P2 and $19/23/5/2\%$ on P4 for N, W, R and RB. Cost was about $1.9.

Table 8 gives the four texts under that rule.

F THE FINAL-ANSWER AUDIT

Table 3's tool-channel rows count calls, and a failure reported in the final answer instead does not appear in them. So we read every final answer of the N and RB branches in the two permanent runs

Table 8: The c2 replay: the same four texts on the 87 permanent states a no-request gpt-4.1-mini agent reached in 49 c2 tasks. A pass here requires *switching* to an alternative tool, the opposite of c1's rule, so these pass rates are not comparable with the c1 rates of Table 1. The ordering inverts: the text that ranks first on c1 ranks last here, because passing c2 requires successful use of an alternative tool (Appendix E).

Text	Benchmark pass	Switched to an alternative	Repeated the failed tool
N	32.2%	83.9%	4.6%
W	**41.4%**	83.9%	4.6%
R	28.7%	77.0%	71.3%
RB	**17.2%**	**56.3%**	85.1%

Table 9: The branches of Table 1 under all four rules, on `C1_task_001–C1_task_040`. *Strict* and *data flow* repeat Table 1; *semantic* forbids only the tools this section's adjudication judges the failure to block — those labels are about requested sub-tasks, but the rule, like the others, still keys on tool names; *none* forbids nothing and is a diagnostic endpoint rather than a candidate evaluator. The attenuation rows are the paired difference from the strict column, positive when the alternative rule credits the message with less. The warning-only arm is in Table 11. Intervals are task-clustered bootstrap on the paired difference.

Model	Text	Strict	Data flow	Semantic	None
Haiku 4.5	N: nothing	10.5%	57.9%	46.1%	89.5%
	R: + retry request	18.4%	60.5%	50.0%	88.2%
	RB: + stop clause	26.3%	67.1%	57.9%	89.5%
	RB−N	+15.8	+9.2	+11.8	+0.0
	[95%]	[6.6, 26.3]	[0.0, 19.7]	[1.3, 22.4]	[−5.3, 6.6]
	attenuation A	—	**+6.6**	**+3.9**	+15.8
	[95%]		[**1.3**, **11.8**]	[**0.0**, **9.2**]	[7.9, 25.0]
	RB−R	+7.9	+6.6	+7.9	+1.3
	[95%]	[2.6, 15.8]	[1.3, 13.2]	[2.6, 15.8]	[0.0, 3.9]
gpt-4.1-mini	N: nothing	26.7%	61.3%	53.3%	86.7%
	R: + retry request	30.7%	64.0%	61.3%	86.7%
	RB: + stop clause	60.0%	78.7%	77.3%	89.3%
	RB−N	+33.3	+17.3	+24.0	+2.7
	[95%]	[23.7, 43.4]	[8.1, 27.0]	[14.5, 33.8]	[0.0, 8.1]
	attenuation A	—	**+16.0**	**+9.3**	+30.7
	[95%]		[**8.0**, **25.0**]	[**2.7**, **17.3**]	[21.1, 40.3]
	RB−R	+29.3	+14.7	+16.0	+2.7
	[95%]	[19.7, 39.5]	[6.6, 24.0]	[7.9, 25.4]	[0.0, 8.1]

— 151 states, 302 answers, no sampling — and coded each one by hand. The rubric, the predictions and the interpretation rules were written and released before the systematic coding of all 302; a subset had been inspected earlier, namely the five states an earlier version of this paper quoted and the answers hand-checked when three keyword classifiers were rejected. That rejection is why this is done by reading.

Rubric. Each answer gets three independent binary codes. **F**: the answer states that the failed step did not succeed — naming the step is not required, and saying nothing about it is **No**. **A**: the answer states that the step's requested outcome was achieved; under P4, reporting the corrupted return as a result counts, unless the answer flags the value as corrupted. **P**: the answer says of at least one *other* sub-task that it was done, or that it was not performed. The rubric allows F and A together, for an answer that reports the failure in one sentence and asserts the outcome in another.

Result. Intervals are the same task-clustered bootstrap used in the body.

	Haiku 4.5 (76 states)			gpt-4.1-mini (75 states)		
Code	N	RB	RB−N	N	RB	RB−N
F, reports the failure	56.6%	73.7%	+17.1 [7.9, 26.3]	60.0%	78.7%	+18.7 [10.7, 26.7]
A, asserts the outcome	42.1%	26.3%	−15.8 [−25.0, −6.6]	36.0%	20.0%	−16.0 [−24.0, −9.2]
P, delimits other work	56.6%	69.7%	+13.2 [5.3, 22.4]	56.0%	66.7%	+10.7 [1.3, 20.5]

Which channel carried the report. Channel-by-channel rates do not say whether one channel makes up for the other, so we crossed the two labels state by state, with the tool channel restricted to the calls to adjudicated-dependent tools that this paper audits (`scripts/channel_crosstab.py`).

States	both channels	tool only	final answer only	neither
Haiku 4.5, N	6	0	37	33
Haiku 4.5, RB	3	0	53	20
gpt-4.1-mini, N	2	0	43	30
gpt-4.1-mini, RB	0	0	59	16

The *tool only* column is empty in all four rows: within each arm, every audited tool-channel failure report came with a final-answer failure report as well. That is a statement about the two channels inside an arm, not about what happened to individual states between arms, so we also crossed the two arms directly, taking a state as reported when either audited channel carries a failure report.

States, N → RB	Haiku 4.5	gpt-4.1-mini
reported under both texts	42	44
gains a report under RB	14	15
loses the report it had under N	**1**	**1**
reported under neither	19	15
total	76	75

The two that lose it are `C1_task_009` under P2 on Haiku, where "Unable to complete, the `compute_trade_amount` service returned an error" under N becomes "All tasks completed successfully" under RB with the amount the failed tool was asked for — that branch still fails the strict rule, for continuing rather than for misreporting — and `C1_task_025` under P2 on gpt-4.1-mini, where `convert_datetime` fails, N says so, and RB states the converted expiry time as fact. So the aggregate improves and the states reporting in neither channel fall from 33 to 20 and from 30 to 16, but not every state keeps its report. None of this is evidence of substitution: nothing here shows a report moving from one channel to the other.

What was predicted, and what happened. Three predictions were fixed before the systematic coding. The first, that the final answer reports the failure in at least 70% of no-message branches, is *wrong*: it is 56.6% and 60.0%, so the final answer does not always tell the user that the step failed. The second, that assertions fall under the full message, holds. For the third we predicted the interval on F's RB−N would contain zero, and committed to withdrawing the body's reading if it excluded zero *downward*, which would mean the message suppresses failure reporting in both channels at once. It excludes zero upward instead.

Limits, and one thing the coding does not support. F and A came out near complements: of the 302 answers none is coded both and only five neither, so they are two views of one two-class split, not two independent measurements, and should not be presented as separate evidence. The coding is by one author with no independent re-rating — unlike the dependence and call-content adjudications, which are by one author and one language model. Eleven states carry a `borderline` marker in the released file. Seven of those answers are the case a rater could reasonably read the other way: all are P4, all report the step as done while warning that the returned value is corrupted, and the pre-registered rule codes a warning about the value as cancelling the claim (A = No). A caution about the value and a reservation about the outcome are not the same thing. We keep the pre-registered reading because it was fixed in advance, and report the alternative beside it. Reading a warning as

compatible with a success claim instead recodes two Haiku N answers, three Haiku RB, one gpt N and one gpt RB, and moves A's RB−N from -15.8 $[-25.0, -6.6]$ to -14.5 $[-23.7, -6.6]$ on Haiku and leaves it at -16.0 $[-23.7, -9.2]$ on gpt-4.1-mini. Recoding the other four borderlines too, where the agent computed a correct conversion itself after the tool failed, gives -15.8 and -17.3 $[-25.3, -9.5]$. The direction and the exclusion of zero survive all three readings. The P rule was ambiguous as written and was clarified once during coding — an answer counts only if it says something about a sub-task other than the failed one — then applied uniformly; that raised P on fourteen answers, and F and A were not revisited.

G THE OTHER SIXTY C1 TASKS

The main text uses the first forty c1 tasks by identifier. That is a fixed prefix rather than a random sample, so the size of the rule effect it reports need not hold on the rest. This appendix replicates the as-run against data-flow contrast on `C1_task_041`–`C1_task_100`, the sixty the paper does not otherwise use, under a pre-registration written before the run (released with the artifact) that fixed the hypotheses, the invalidation rules and what to do with either outcome.

Design. gpt-4.1-mini, modes P2 and P4, judge and prompts unchanged, texts N and RB only, one no-request baseline episode per task-mode followed by same-state branches exactly as in Section 3. The semantic rule is not computed: it needs adjudication labels, none exist for these tasks, and producing them would again be one author and one language model. The paper's headline contrast does not depend on those labels, so this replication targets it directly.

What ran. Of the 120 baseline episodes, none recorded a provider error as a final answer and all ended in the agent's own final answer. Six tasks — `044`, `048`, `052`, `067`, `078`, `087` — never called the fault-injected tool under either mode, so no faulted state exists to branch from and they are excluded, as in the main run. The remaining 54 tasks give 108 states and 216 branches, with no replay mismatch, no provider error and none at the round limit.

Result. Under the as-run rule the message helps: N passes 36.1% and RB 47.2%, RB−N $= +11.1$ $[3.7, 19.4]$, the interval excluding zero as predicted. Under the data-flow rule the levels rise, 48.1% and 58.3%, but the gap barely moves: RB−N $= +10.2$ $[3.7, 17.6]$. The attenuation A of Section 4 is $+0.9$ $[-1.9, +4.6]$ — an interval containing zero, and disjoint from the 16.0 $[8.0, 25.0]$ measured on the first forty. The reclassification is correspondingly symmetric: relaxing the rule turns an as-run failure into a pass in 13 of 108 states under N and 12 under RB, against a two-to-one split on the first forty.

Why the two rules nearly agree here. The reduction in the measured effect is not a separate quantity from the reclassification above: it is the rescue count under N minus the rescue count under RB. Here that is 13 against 12 of 108, a difference of one state, which is the $+0.9$; on the first forty it is 26 against 14 of 75, a difference of twelve, which is the 16.0. What needs explaining is the asymmetry, not the shift in levels.

One quantity consistent with it is how much of an episode the relaxed rule can excuse at all, since it excuses a later call only when `_compute_dependent_tools` does *not* mark that tool as consuming the failed output. On these tasks it marks more: the mean dependent-set size per state is 0.91 tools against 0.53 on the first forty, and of the states that made a forbidden call it excuses all of them in 15 of 57 under N and 13 of 44 under RB, where on the first forty it excused 30 of 50 and 15 of 24. That is a description of these two task sets and not a tested cause: how many forbidden calls each arm makes, and which tools they go to, would have to be held fixed to isolate it.

What this changes. The pre-registration committed in advance to reporting this outcome and narrowing rather than re-running or re-slicing, and we have done neither. Tables 1, 2, 3 and 11 are unchanged. The replication supports the paper's main claim — that the same trajectories receive different credit under rules that differ only in the forbidden set, so a benchmark shipping both rules reports a rule-dependent number — with the two rules moving the levels by 12 and 11 points on a task set where they leave the *contrast* intact. It does not support the magnitude: how much of the measured benefit the choice of rule accounts for varies with the task set, in a way the dependency

density above is consistent with but does not establish, so the 6.6 and 16.0 of Table 1 are measured on `C1_task_001`–`C1_task_040` and not a benchmark-level constant. The recommendation follows either way: which rule an evaluator uses cannot be assumed to matter little, nor to matter the same amount on a different task set, so it has to be stated and reported.

H SAME-STATE BRANCH: DETAILS AND THE ONE-STEP VERSION

The *none* column. With no later call forbidden, the remaining conditions leave RB−N at 0.0 and 2.7 points, on intervals whose upper ends are 6.6 and 8.1. A near-zero mean can hide offsetting changes, and here it does: under *none* on Haiku two states pass under N and fail under RB and two do the reverse, which is how +0.0 arises. The column bounds what the criterion's other conditions contribute; it is not a claim that behaviour was unchanged.

What each fault regime's pass requires, and the runs behind the figures. A transient pass requires both the repeat call and completion of the required tools, so it is not credit for an action alone; what it does not check separately is the final answer. A permanent pass is the trace check of Section 3. The transient-fault and clean-branch figures quoted in Section 3 come from the earlier Haiku execution `exp15_branch_run2` (Table 10); every permanent-fault figure in Tables 1–3 comes from `exp15_perm_regen` and `exp17_branch`, and the two are never mixed. On transient faults the Haiku rise from the warning arm is +65.8pp $[+52.6, +77.6]$ and from no text 22.4% to 81.6%, and from the warning arm 15.8% to 81.6%; the body quotes the second, since the warning arm is the transient reference both models ran. gpt-4.1-mini ran W and R on transient faults and all four texts on permanent ones, so no transient no-text arm exists for it. Under RB the 69.8% of clean branches carrying a repeat call cost +2.2k continuation tokens, while completion on those branches is 88.4% under all four texts with no discordant pair — which describes this sample rather than supporting a non-inferiority claim.

Quantities held fixed by the replay. Identical across a prefix's branches are the transcript, the call count and the execution context. Manipulated are the result at the branch point and, for a clean branch, the fault regime of the rest of the episode, which runs on a fault-free engine; that comparison is between a faulty world and a fault-free one, not between worlds differing at one call. Each model's branches are conditional on the states its own agent reached, so the two are not evaluated on common states; early against late delivery was run on Haiku only, the specific-cue comparison on gpt-4.1-mini only.

Prefixes. Of the 160 no-request episodes in P1–P4, 152 reached the fault-injected tool; the other eight did not call it and are excluded. Clean branches are identical across modes of a task whose prefixes coincide up to the fault-injected tool call, so they are run once per distinct (task, call sequence): 43 prefixes over 40 tasks (three tasks have two). The pre-registration note (released) records the predictions, the analysis plan, the harness check on two prefixes, the deduplication decision made after that check and before the run, and the invalidation of a first full run in which 166 of 780 branches ended in provider errors recorded as final answers; the run reported in Section 3 is the complete rerun, with zero such branches, zero prefix mismatches and zero round-limit hits.

Run-to-run variability. The 614 branches of the first run that did not end in a provider error are an independent execution of the same conditions, and give a direct estimate of how much these measurements move between runs at temperature 0: the two runs agree on whether the agent retried in 95.4% of them, on the judge's verdict in 97.1%, and on whether it proceeded past the fault in 98.9%. On the permanent-fault branches present in both runs (60 per condition) the contrast of Section 3 reproduces: proceeding past the fault falls from 80.0% under R to 70.0% under RB in the first run and from 80.0% to 71.7% in the second, with success rising from 11.7% to 23.3% and from 13.3% to 20.0%. The effect is therefore not an artifact of a single execution, though both runs share the same prefixes.

Figure 3 shows all three fault regimes arm by arm.

Secondary contrasts of the branch experiment. These are computed on the recorded run (`exp15_branch_run2`), the same run as Table 10, so the permanent rows here differ slightly from

Table 10: Same-state branches by fault persistence and text. "Success" is the benchmark's pass, which credits completion after the retry on transient faults and stopping after the fault on permanent ones (Section 3); the two blocks therefore measure different things and should not be compared with each other. "Proceeds unresolved": another tool called before the fault was resolved (before the clean retry result on transient faults; anywhere after the fault-injected tool on permanent ones). Tokens are the continuation's input plus output, in thousands.

Branch	Text	Retry	Success	Proceeds unresolved	Tokens (k)
Transient (n=76)	N	27.6	22.4	65.8	7.0
	W	19.7	15.8	72.4	6.9
	R	94.7	81.6	5.3	9.2
	RB	100.0	88.2	0.0	9.5
Permanent (n=76)	N	3.9	9.2	88.2	6.3
	W	5.3	14.5	84.2	6.3
	R	94.7	14.5	80.3	8.7
	RB	98.7	22.4	71.1	8.5
Clean (n=43)	N	0.0	88.4	—	6.3
	W	0.0	88.4	—	6.3
	R	39.5	88.4	—	7.6
	RB	69.8	88.4	—	8.5

Table 11: Permanent-fault branches by text, from the runs every main-text number uses (`exp15_perm_regen`, `exp17_branch`). *Repeats* is a second call to the fault-injected tool; *continues* is a call to any other tool after it, which the strict rule forbids; *pass* is that rule's verdict. Across the two models the whole message lowers disallowed continuation by 18.4 and 34.7 points, of which the conditional continuation/termination clause alone (R to RB) contributes 9.2 and 28.0. Transient-fault and clean branches are tabulated in Appendix H, which also reports an earlier execution of the same conditions.

Model	Text	Repeats the call	Continues to another tool	Pass (strict rule)
Haiku 4.5 (76 states)	N	6.6%	86.8%	10.5%
	W	5.3%	84.2%	14.5%
	R	92.1%	77.6%	18.4%
	RB	98.7%	68.4%	26.3%
gpt-4.1-mini (75 states)	N	5.3%	66.7%	26.7%
	W	10.7%	68.0%	25.3%
	R	97.3%	60.0%	30.7%
	RB	98.7%	32.0%	60.0%

Table 11. Beside the three pre-registered primary contrasts, the following are secondary, task-paired with task-clustered 95% intervals:

Contrast	pp [95% interval]
Transient success, W−N	−6.6 [−14.5, +1.3]
Transient retry, R−W	+75.0 [+64.5, +84.2]
Transient success, RB−R	+6.6 [+1.3, +11.8]
Permanent success, R−W	0.0 [−5.3, +5.3]
Permanent success, RB−R	+7.9 [+1.3, +15.8]
Permanent retry, RB−R	+3.9 [0.0, +9.2]
Clean retry, RB−R	+30.2 [+15.6, +46.3]
Clean tokens (k), RB−R	+0.94 [+0.42, +1.50]

Timing of delivery. RB attaches its clause to the first faulty result, before the agent has retried, and also raises retries slightly, so its effect mixes the instruction with the retry it induces. A pre-registered run separates them by assigning at the first result, so that retrying is an outcome rather than a selection rule, and attaching a follow-up to the agent's own repeat call: INFO states only that the result is unchanged, STOP adds the conditional stop clause. Against INFO, STOP lowers disallowed continua-

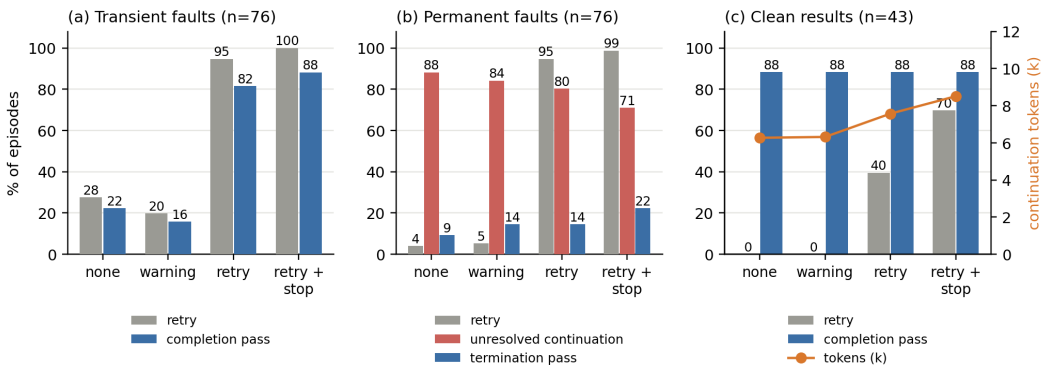


Figure 3: Same-state branches, Haiku 4.5, from the earlier execution `exp15_branch_run2`; its permanent-fault rates differ slightly from the regenerated run behind Tables 1 and 11 and are not mixed with them. From the state at which a no-request agent reached the faulty tool call, the result is branched and one text attached: nothing (N), a warning (W), a retry request (R) or a retry request with a conditional stop instruction (RB). All three panels use the same 0–100 axis, but a pass means something different in each: (a) transient faults — repeating the call, and a pass that requires completion after that repeat; (b) permanent faults — repeating the call, calling some other tool while the fault is still unresolved (permanent faults never resolve, so any other call counts), and a pass that requires appropriate termination; (c) clean results from the same prefixes — repeating the call, completion under the P0 judge, and continuation tokens on the right axis.

tion by 11.8pp ($[-19.7, -5.3]$); INFO itself is indistinguishable from R (-1.3pp, $[-6.6, +2.6]$). The late clause's effect is not distinguishable from the corresponding early one (-11.8 against -9.2pp; difference -2.6pp, $[-9.2, +3.9]$), so we do not identify an effect of timing — a null result also reported by Outcome Monitors, and one our sample does not resolve any better. An earlier version of this comparison conditioned on states where the agent had already retried, which selects on a post-treatment outcome; it reported a larger late effect that this design does not reproduce (Appendix H).

Naming the violation, without asking for anything. A cue being present does not mean the agent read it, so a pre-registered run has the runtime point at one: from the 68 gpt-4.1-mini permanent states where a violation can be named out of what the agent received (the classifier of Appendix A; 7 states with no nameable violation and 5 that never reach the fault are excluded), a text states that violation and asks for nothing — for example "`AUTOMATED CHECK: this call was made with flight_number='CA1234', but the result reports flight_number='CA4321'.`" All 68 texts were read by hand before the run to confirm each states only what was sent and what came back. On implicit faults (n=30) CUE lowers disallowed continuation from the warning arm's 83.3% to 70.0% — the no-text arm is 93.3% — which we cannot resolve (-13.3pp, $[-30.0, +3.3]$), and RB beats it by -20.0pp ($[-33.3, -6.7]$). On explicit faults (n=38) the effect is -28.9pp ($[-44.7, -15.8]$), 11 discordant pairs to 0, where we had pre-registered little because the error string was already verbatim in the result; there RB and CUE are indistinguishable (-7.9pp, $[-21.1, +5.3]$). Retries stay at the warning's level under CUE — 6.7% and 2.6% against the warning's 6.7% and 10.5%, and 96.7% and 100% under RB — so it is not read as an instruction to call again; declared aborts rise instead, 42.1% to 76.3% on explicit faults. All 68 branches ran to the agent's own final answer with no round limit, replay mismatch or provider error.

Delivery timing, and a withdrawn design. The timing arms run from the same 76 permanent-fault states. Both carry the R text at the first faulty result, and the follow-up is attached inside the continuation loop to the agent's own repeat call, so an agent that never repeats receives none; that happened in 5 of 76 per arm. All 152 branches ran to a final answer with no round limit, replay mismatch or provider error, and retry rates are 93.4% in both arms against 94.7% under R, inside the 10pp band fixed in advance. An earlier attempt took the 72 states in which the agent had *already* retried under R and attached the follow-up there; retrying is an outcome of the first text, so that conditions on a post-treatment variable. It gave a larger late effect (-16.7pp on proceeding,

$[-27.7, -7.0]$), which the design above does not reproduce. We report the withdrawn numbers so the two designs can be compared, not as evidence of timing.

The one-step variant. Before the full-episode branches, the same 152 prefixes were branched into faulty or clean results with and without the V0 text, with the model asked for one next action (608 calls). Without a request the next action retries on 16.4% of faulty results and $0/152$ clean ones; with V0, 97.4% and 75.7%. The request adds $+80.9$pp on faulty results ($[+74.3, +86.8]$) and $+75.7$pp on clean ones ($[+61.8, +88.2]$), a difference of $+5.3$pp ($[-7.2, +18.4]$) that is not distinguishable from zero. The replay reproduced the logged next action in 93.4% of no-request cases (`scripts/exp13_analysis.py`).

I RUN INVENTORY

Table 12 lists every run reported in the paper and the section that uses it.

Table 12: Every run this paper draws a number from, including the appendices; n is tasks per run. Earlier exploratory runs that no current result uses are in the repository's run history, not here.

Run	Agent, tasks	Arms	Question
Whole-episode runs: a request policy acts from the start of the episode. Used here only for the taxonomy and value-use counts of Appendix A.			
Formulation run	Haiku 4.5, c1, n=40	always with V0 / V1 / V2; gated; none	wording of the request
Second-model wording run	gpt-4.1-mini, c1, n=40	always with V0 / V2	pre-registered replication
Closing-clause run	Haiku 4.5, c1, n=40	always with V0cmp / V0stop	pre-registered; clause only
Same-state branch runs: a logged episode is replayed to the fault-injected call and the result is branched there.			
Same-state branch, one step	Haiku 4.5, c1, 152 prefixes	faulty / clean $\times$ no request / V0, next action only	request effect vs. fault status
Same-state branch, full episode	Haiku 4.5, c1, 152 faulty + 43 clean prefixes	N / W / R / RB, run to completion, judged per branch	failure step vs. text
Timing branch	Haiku 4.5, c1, 76 permanent states	R first, then INFO / STOP on the agent's own repeat call	when the instruction arrives
Specific-cue branch	gpt-4.1-mini, c1, 68 permanent states with a nameable violation	CUE: names one violation, asks for nothing	pointing at the fault vs. instructing
Post-retry branch (withdrawn)	Haiku 4.5, c1, 72 failed-retry states	nothing / continue / stop-and-report	selects on a post-treatment outcome
Second-model branch	gpt-4.1-mini, c1, 75 permanent + 75 transient	permanent: N / W / R / RB; transient: W / R; no clean branches (450)	pre-registered replication
Permanent-branch regeneration	Haiku 4.5, c1, 76 permanent states	N / W / R / RB, conversations retained (`exp15_perm_regen`)	the run Sections 4–6 re-score
c2 replication	gpt-4.1-mini, c2, 49 tasks, P0 / P2 / P4	baseline and the four texts (146 baseline episodes)	a different scoring rule
Off-prefix replication	gpt-4.1-mini, c1, tasks 041–100, P2 / P4	N / RB from 108 permanent states (120 baseline episodes; `exp22_offprefix`)	does the rule gap hold off the prefix
Stated-policy runs: the system prompt's fault clause is replaced and the episode is run from the start.			
Policy runs	gpt-4.1-mini, c1, 40 tasks, P2 and P4	ATOMIC / SALVAGE, no runtime text (80 each; `results_exp21_*`)	does the score follow the stated policy

Table 13: Which script recomputes which result, from the stored per-call records and evaluations.

Script	Recomputes
`rules_by_text.py`	Tables 1 and 9: every text under every continuation rule, both contrasts and their intervals
`structural_dependency.py`	Table 2: the benchmark's data-flow labels on the adjudicated relations
`exp22_offprefix_rules.py`	Table 1, third column, and Appendix G: the off-prefix replication under both rules
`final_answer_full_audit.py`	Table 3, last two rows, and Appendix F: the hand-coded final-answer audit
`channel_crosstab.py`	Appendix F: the two audited channels crossed within and between arms
`verify_paper_numbers.py`	checks the reported numbers in this manuscript against the scripts above
`semantic_label_sensitivity.py`	the alternative-label recomputation of the semantic column
`valid_independent_work.py`	Table 3: structured-field agreement on adjudicated-independent sub-tasks
`subtask_outcomes.py`	the per-sub-task verdicts the previous two build on
`structured_field_by_class.py`	Section 6: both classes of later sub-task under the structured-field bar, and the difference between their contrasts
`adjudicated_by_class.py`	the same split on call counts, quoted in Appendix B
`content_aware_rescoring.py`	the semantic column with failure-reporting calls excused (Section 5)
`exp21_policy_analysis.py`	Table 7 and the state-level decomposition
`notification_content_audit.py`	the classification of what the disputed calls' arguments said
`build_adjudication_sheets.py`	the adjudication sheets, with the automatic verdicts kept separately
`exp15_full_branch.py`, `exp15_analysis.py`	the same-state branches; the second-model arm reuses it with the model, source arm and texts set by environment variables
`continuation_taxonomy.py`	stopping, independent continuation and value reuse
`failure_taxonomy.py`, `fault_discriminability_v2.py`	the per-episode failure classes and which faults carry a visible cue
`exp20_c2_analysis.py`	the c2 replication

J PROMPTS AND REQUEST TEXTS

Forecast elicitation (verbatim). Issued out of band; the response never enters the acting agent's transcript as a forecast, though under the evidence text a copy is placed inside the request.

```
Transcript of an agent working on a task:
{transcript}
The agent is about to call tool `{tool}` with arguments
{arguments}.
Before seeing the result, forecast what this call will
return.  In 20 tokens or fewer, state the salient
fields/values you expect (status, key numbers, ids,
or error).  If you cannot predict, output exactly:
unpredictable.  Output only the forecast.
```

Agent system prompt in the fault modes (verbatim). Used unchanged from ToolMaze; P0 uses that suite's plain prompt instead. Every request text below is appended to a tool result inside an episode running under this prompt.

```
You are an autonomous AI assistant that executes tool-based
workflows.  Tools may occasionally fail or return unexpected
data.
```

```
TASK COMPLETION
- The user query may describe multiple sub-tasks.  You must
complete ALL of them --- do NOT skip any sub-tasks.
- Every fact, data retrieval, or state change must come from
a tool call.  Never substitute a tool result with internal
knowledge, and never report a step as completed unless you
actually called the corresponding tool and received its
result in this conversation.
- Sub-tasks may be accomplishable through different tool
combinations.

FAULT HANDLING
- When evaluating tool outputs, consider whether the
returned data is coherent and usable for the next step ---
fields should be present, values should be semantically
reasonable, and types should match what the workflow
requires.
- If you determine that a required step cannot be
accomplished by any available tool or approach, stop the
workflow and clearly report which step failed and why,
rather than proceeding with incomplete results.

FINAL ANSWER FORMAT
- Be CONCISE and DIRECT. No emojis, markdown formatting, or
bullet points.
- If SUCCESSFUL: State only essential facts and requested
results (e.g., confirmation codes).  Omit internal technical
details.
- If HALTED: State clearly that the task was aborted and the
reason.  Do not apologize excessively.
```

Request texts (verbatim).

evidence-carrying: AUTOMATED CHECK: before this call, the expected result was: "{forecast}". The actual result diverges from that expectation. If the actual result is inconsistent with the task context, call this exact tool one more time with the same arguments to confirm, then continue with the confirmed value.

V0: AUTOMATED CHECK: this result may be unreliable. Please call this exact tool one more time with the same arguments to confirm the result, then continue the task with the confirmed value.

V1: VERIFICATION REQUIRED: re-run this tool call with identical arguments and use the second result.

V2: Note from the runtime: results from this tool are not always stable. Before relying on this value, please repeat the identical call once and proceed with what it returns.

continue control: AUTOMATED CHECK: this result has been recorded for review. No action is required; continue the task with this value.

V0cmp (closing-clause run): AUTOMATED CHECK: this result may be unreliable. Please call this exact tool one more time with the same arguments to confirm the result, then compare the two results and continue with the one you judge correct.

V0stop (closing-clause run): AUTOMATED CHECK: this result may be unreliable. Please call this exact tool one more time with the same arguments to confirm the result; if the second result is also faulty, stop and report it rather than continuing.

V0, V0cmp and V0stop share the same warning and retry request and differ only in the clause that follows, so the run isolates what that clause adds.

Follow-up texts attached after the retry (verbatim). Used in the timing experiment (Appendix H); the first result carries the R text in both arms, and the text below is attached to the result of the agent's own repeat call, if it makes one.

> **INFO:** AUTOMATED CHECK: this result is unchanged from the previous call.
>
> **STOP:** INFO + If it still shows the same problem, stop and report which step failed rather than continuing.

K SCORING IS DETERMINISTIC

The two permanent-fault trace checks. At commit `ef0798a`, `evaluation/core/judge.py` defines two checks for P2 and P4. `_check_p2_p4_stop_after_victim` (L1193) locates the perturbed step, requires every earlier required tool to appear in the call record and the victim itself to be called, forbids *any* non-victim call occurring after the victim's first appearance, and, when the victim is the last required step, requires an explicit abort statement in the final answer. `_check_p2_p4_trace_dependency_aware` (L1360) shares the first two conditions but forbids only the tools that `_compute_dependent_tools` (L1307) derives from the task template's `parameter_bindings` as transitively consuming the victim's output, recording that "independent tools are allowed" (L1430); it tests membership in the whole trace rather than the part after the victim, and has no abort condition. `judge()` dispatches c1 P2 and P4 to `_check_p2_legacy` (L76) and `_check_p4_legacy` (L80), both of which call the first; the second has no call site in the file.

Because the two differ in three conditions, Table 1 varies only the forbidden set and holds the others at the as-run check's values. Applying the second function unmodified gives 60.5% under N and 67.1% under RB on Haiku and 66.7% and 80.0% on gpt-4.1-mini, against 57.9/67.1 and 61.3/78.7 for the controlled data-flow column, so the position and abort conditions move these rates by at most a few points.

The trace check against the perturbation recovery rate. At commit `ef0798a`, `judge.py` passes a c1 transient episode when the fault-injected tool is called at least twice *and* the required tools then complete, and a c1 permanent episode when no other tool is called after the fault-injected one. `metrics.py`, which feeds the perturbation recovery rate (PRR), instead counts a transient hit recovered when that tool later succeeds un-perturbed whatever happens to the rest of the task, and a permanent hit recovered when no *successful* call follows. Both of the following are recovered under PRR and failed under the trace check: an episode whose repeat call returns a clean, un-perturbed result but which then fails to finish the required tools, and an episode that calls another tool after a permanent fault where that call itself fails, leaving no successful call after the fault. Recomputing both verdicts on our logged c1 episodes gives

	no request (152)		always V0, Haiku (156)		always V0, gpt (151)	
	PRR rec.	PRR not	PRR rec.	PRR not	PRR rec.	PRR not
trace pass	27	0	66	6	82	3
trace fail	10	115	9	75	14	52

so they agree on 93.4, 90.4 and 88.7% of episodes, and the disagreements are mostly PRR crediting what the trace check fails (10, 9 and 14 episodes against 0, 6 and 3 the other way). This paper analyses the episode judgement behind task success. PRR is computed separately, and the difference between the two is reported here rather than carried into our conclusions.

The run configs carry a `judge.model` field, which invites the reading that episodes are graded by a language model from the agent's own family. `JudgeSystem.__init__` accepts an `llm_client`, stores it, and no scoring path reads it: of the 53 methods on that class, `__init__` is the only one that

mentions it, and every complexity class dispatches to a deterministic trace check. In the 1,200 stored evaluations every record carries a `trace_check`; `pass` equals `trace_check.match` in 1,168 of them, and the remaining 32 fail a further deterministic requirement. The mode rules for c1, read from `judge.py` (other topologies admit alternative paths and are scored differently): P1 and P3 pass only if the fault-injected tool is called at least twice and the required tools then complete; P2 and P4 pass only if the agent stops after the fault-injected tool, and any other tool called after it fails the episode. The script that checks both halves is released. Consequently, cross-agent comparisons cannot be confounded by a model judge's leniency, though whether the trace criteria suit each agent's recovery style is a separate question. The retry measure and the outcome measure are computed by different code paths, though for transient faults they are not independent, since the benchmark's own pass condition requires a retry that the agent performed.

L SCOPE NOTES

The stop rule as a policy choice. An evaluation may legitimately test compliance with a stated rule, and ToolMaze states this one; our claim is only that its output should not be read as an error-propagation rate, nor as identifying which component of recovery competence changed. The choice between halting the whole episode and permitting the parts the failure did not block is a policy choice; Section 4 quantifies how much the estimated effect of an instruction depends on it, and does not settle it.

Round limits and non-retries. Every episode in every whole-episode run ends with the agent's own final answer, inside the round budget; none is cut off at the round limit. When a request is attached and the agent does not retry, it therefore had budget left to act; what it was doing instead is not something the round count can say. The same holds of the same-state branches (Section 3), which are also run to the agent's final answer.

M ANALYSIS VALIDATION AND CORRECTIONS

Five readings from exploratory analyses were corrected before the results reported here were fixed. They are of four kinds: an effect that disappeared once the design stopped conditioning on an outcome, two aggregation errors in our own code, an over-broad reading of a classification, and a metric read as the construct it approximates. The reusable part is the validation procedure in the third column; each was then applied to every result of its kind, and the numbers reported in this paper are those that pass. The table also shows which body results were affected: all five concern the diagnostic and dependence analyses, and none touches the re-scoring of Table 1, which is computed from the stored branch traces under the two rules and is unchanged by any of them.

Two facts about the provenance of the runs belong here too: the Haiku permanent traces were regenerated after a file-naming error overwrote the originals (Appendix A compares the two), and a first branch run was invalidated when 166 of 780 branches recorded provider errors as final answers, then rerun in full (Appendix H).

Earlier reading	What it missed	Validation applied	Reported conclusion
A stop clause is twice as effective after the failed repeat call	Conditioned on states where the agent had already repeated, an outcome of the first text	Re-running the design so the repeat call is an outcome, not a selection rule	Late -11.8pp against early -9.2pp; difference -2.6 $[-9.2, +3.9]$, no timing effect identified
One credited episode in ten carries the stale value	Counted a perturbed value found downstream without checking for the clean one, or for values that appear for other reasons	Reading the matched text for every flagged episode	No stale-only episode was identified; one mixed episode retained an incorrect field
Continuing after a permanent fault propagates the failure	Matched JSON field names as well as values, and never asked whether the task had a downstream consumer	Inspecting the matched value in one episode; then measuring opportunity from the fault-free run	The value-reuse test identifies a potential downstream consumer in 6 of 76 states; the score is not a propagation rate
98% of implicit faults are discriminable	Classified using the clean output, which the agent never receives	Asking which classifier inputs the agent actually had	86% (65 of 76) on agent-visible cues; the dominant cue is a contradicted argument
A conditional stop instruction costs 19.8 points of independent downstream work	"Independent" meant only that no value of the failed call was reused; a sub-task can depend on the failed step with no value in common	An adjudication by two raters, blind to condition, under a rubric fixed before any label was assigned, of all 62 (task, later tool) relations	-9.7pp $[-22.2, 0.0]$ on adjudicated-independent sub-tasks against -17.9pp on dependent ones, both on the structured-field bar; the reuse test found 4 of 29 dependencies (Section 5)