

Feedback That Backfires: Why Small Language Model Agents Repeat the Call They Just Watched Fail

Esmail Gumaan*

Faculty of Computer Science and Mathematics
University of Passau, Germany

Code, data and every number in this paper:

<https://github.com/Esmail-ibrahem/feedback-that-backfires>

Abstract

Agent harnesses record a failed tool call and its error message in the transcript and ask the model to continue, on the reasonable assumption that the error is corrective information. We measure whether it is. Defining the *corrective gain* of a failure record as the change in log-probability of re-emitting the action that just failed, we find the gain is *negative* for every instruction-tuned model we tested (6 checkpoints, 135M–1.7B, 4 families) in two environments: simulated tool calling and MBPP program repair with real interpreter output. Normalised by action length the effect has the same size in both, about -1.03 nats per action token, a factor of 2.8 in the odds of each token. It holds on 90%–100% of individual items, not merely on average. Over a fixed candidate set the normalised probability of repeating the failed call rises from 0.06 to 0.54, and greedy decoding reproduces the failed call token for token on 19% of items after the failure versus 0% before it. Counterfactual observations, in which the same call is paired with a failure message, a success message, or a valence-free acknowledgement, separate two competing effects. The failed call’s *surface form* accounts for 83% of the damage; the semantic contribution of marking it as failed is small and its sign is not even consistent across environments. The problem is therefore in the harness, not in the model’s grasp of error messages, and that predicts which remedies can work. Replacing the verbatim call with a runtime-generated description of the failure removes 76% of the inversion at no token cost, and making previously-failed strings unreachable at the decoder acts on the same term. Two remedies that sound right do not act on it at all. An explicit “do not repeat” instruction leaves the measured quantity where it was; and deleting the failed attempt to retry from a clean context, the standard prescription for context contamination, is the worst harness we measured for repetition, because it restores precisely the context that produced the failure. What survives is narrow enough to be useful: the context after a failure must differ from the context before it, and the difference must not be the failed action. The study runs end to end on a CPU and all artefacts are released.

1 Introduction

An agent that calls tools will get some of those calls wrong. What happens next is decided not by the model but by the harness: it appends the failed call to the transcript, appends whatever the runtime printed, and asks the model to continue. ReAct [Yao et al., 2022] introduced this pattern and essentially every agent framework since has kept it, including the ones built for small open-weight models. The reasoning is hard to argue with. The error message is real information, it was produced by ground truth rather than by a critic, and a model that reads it should not try the same thing again.

We measured whether that is what happens. It is not.

Take a decision point in an agent trajectory: a context C , an action $a^\times$ that will fail there, and the error the runtime

returns. Compare the model’s probability of writing $a^\times$ before the attempt with its probability after the failure has been recorded. For every one of the 6 instruction-tuned models we tested, spanning 135M–1.7B parameters across 4 families, that probability goes *up*, by -17.38 nats, or about -1.03 nats per token of the action, which is a factor of 2.8 in the odds of every token. The effect is not an average over a few dramatic cases: it holds on 90% or more of individual items.

Restated on a scale that is easier to hold onto: over a fixed set of four candidate actions, the probability of re-emitting the failed call rises from 0.06 before the attempt to 0.54 after it. Greedy decoding, which is what the agent would actually have written next, reproduces the failed call token for token on 19% of items after the failure, against 0% before it.

Practitioners know the symptom. Agents get stuck repeating a call that has already failed, and frameworks ship loop detectors and step caps to contain it. The symptom is usually attributed to the model: it is small, it does not really understand the error, a stronger model would recover. Our second result is that this diagnosis is wrong, and it matters because it points at the wrong fix.

Appending $(a^\times, o^\times)$ does two things at once, and they can be separated. It puts the token sequence of the failed call into the context, where the model’s copying machinery can reach it [Olsson et al., 2022, Xu et al., 2022]; and it asserts that the call failed. We can measure each part on its own by scoring the same action under a context whose observation is *valence-free*, meaning the runtime acknowledges the call and says nothing about how it went. The surface-form term accounts for 83% of the effect. The semantic term, the part that would have to be large and negative for “the model does not understand the error” to be the explanation, is small, and its sign is not even consistent between our two environments. On tool calls it is slightly positive, since explaining what went wrong requires naming the tool and the argument again; on program repair it is slightly negative, so a traceback does push the model away from the code that produced it. We stress the second case, because it is the one that fixes the interpretation: these models are not simply failing to read error messages. Whatever reading happens is being outvoted, by a median of 7 to one across model and environment, and by 106 to one at the extreme.

This changes what a fix should look like. If the model misunderstood, the remedy is a better model or a clearer message. If the harm is carried by the presence of a string, then message engineering cannot reach it, and neither can an instruction: telling a model not to repeat a call requires it to represent a negation about a string that is sitting right there in its context, which is the condition under which negative instructions are known to work least well [Castricato et al., 2024, Vrabcová et al., 2025]. What should work instead is structural.

“Structural” turns out to be narrower than it first appears, and the experiment that shows this is the one we did not expect. The obvious structural fix is to delete the failed attempt and retry from a clean context, which is what prior work on context contamination recommends [Yang, 2026]. That is the *worst* harness we tested, multiplying the exact repeat rate by 2.6 without changing task success. The reason is embarrassingly simple once seen: deleting the failure restores the exact context that produced it, and a deterministic policy in an identical context emits an identical action. The requirement is therefore not “remove the failed action” but something more specific: the context after a failure must differ from the context before it, and the difference must not be the failed action.

We test all three. Replacing the verbatim call with a description the runtime generates from its own error metadata, which keeps the diagnosis and drops the token sequence, removes 76% of the inversion and drives the exact greedy

repeat rate to zero, at no cost in generated tokens. A decoder-level ban on previously-failed strings acts on the same term and costs nothing at all. The natural-language prohibition moves the measured quantity slightly the *wrong* way, which is what the negative-instruction literature would predict for a prohibition that has to name the very string it is prohibiting [Castricato et al., 2024, Vrabcová et al., 2025], and in free-running rollouts it does not reduce repetition either, though it does change behaviour in another way we report rather than explain. Section 8 gives all of this in full, including where the interventions do not help.

We are not claiming that execution feedback is useless, or that agent transcripts should be thrown away. The failed call carries diagnostic information a harness genuinely needs. The claim is narrower and, we think, more useful: the standard way of *delivering* that information hands the model a strong reason to repeat itself, and a harness can keep the diagnosis while withholding the reason.

Contributions.

1. **A measurement.** We define the *corrective gain* of a failure record, the change in log-probability of re-emitting the failed action, and show it is negative across 6 models, 4 families and two environments (simulated tool calling and MBPP program repair with real interpreter output). We report it alongside two interpretable readouts, a normalised repeat probability and an exact greedy repeat rate.
2. **A decomposition that identifies the cause.** Counterfactual observations split the gain into a surface-form term and a semantic term. The surface-form term dominates at every scale we tested, and it, not the semantic term, is what improves with model size.
3. **Interventions that follow from the decomposition, and one that does not.** Replacing the verbatim failed call with a runtime-generated description of the failure, and banning previously-failed strings at the decoder, both act on the term that matters; adding a natural-language prohibition does not. We evaluate all three in free-running rollouts and report their cost.
4. **A reproducible, CPU-only artefact.** Every number comes from code in the accompanying repository, which runs end to end without a GPU. Raw per-score records, item sets, seeds and environment metadata are included, and the manuscript’s inline numbers are generated from them rather than typed.

2 Related work

Agents that act, observe, and try again. ReAct [Yao et al., 2022] established the template that dominates practice: interleave a written action with the environment’s observation and let the model condition on the growing transcript.

Reflexion [Shinn et al., 2023], Self-Refine [Madaan et al., 2023] and self-debugging [Chen et al., 2023] all build on it, adding an explicit critique or a test result to the same append-only transcript. MINT [Wang et al., 2023] evaluates multi-turn interaction with tools and language feedback directly and finds feedback helpful in aggregate. The shared premise is that appending the failure is informative; what none of these measure is the counterfactual we study here, namely what the appended *action string* does on its own.

Doubts about self-correction. A parallel line argues that models cannot reliably correct themselves without an external signal [Huang et al., 2023, Valmeekam et al., 2023]. Our result is not a restatement of that. Those papers ask whether a model can *judge* its own output; we hold the judgement fixed, since the environment executes the action and the verdict is ground truth, and ask whether the model’s next-token distribution moves in the direction the verdict implies. The failure we document survives perfect feedback.

Context contamination. The closest prior work is Yang [2026], who formalises the observation that a retry in a context still containing the failed attempt has a higher error rate than a clean one. The Context-Contaminated Restart Model gives closed-form results for success under a budget and fits SWE-bench Verified data with a cascade ratio $\epsilon_1/\epsilon_0 = 7.1$, showing that assuming independent retries overestimates pass@3 by 17.4 points. That work establishes the phenomenon at the level of task statistics and concludes that the remedy is to *clear the context before retrying*. Our contribution is orthogonal and, we think, complementary in a useful way: we ask which *part* of the context does the contaminating, measure it at the token level, and find that it is the failed action’s surface form rather than the error message. That matters practically, because clearing the context discards the diagnosis along with the hazard. In our terms, clean restart is the *drop* harness, and it is a baseline we evaluate rather than an endpoint.

Agents that get stuck. Repetition in agent trajectories is widely reported and usually treated as an engineering defect. Hou et al. [2026] detect unbounded feedback paths by static analysis of agent *source code* across 6,549 repositories, which is complementary to our question: their loops are properties of the program, ours of the policy. Wang et al. [2025] show that agents struggle to formulate alternative plans after an external failure even when the task is guaranteed to remain solvable, which is a behavioural counterpart to what we measure distributionally. Most striking is Zenkri and Brock [2026], who find on a mechanical puzzle that *degrading* an embodied agent’s observations improves success up to $2.85\times$, and trace the gain to fewer repetitive action loops. That is independent evidence that the observation channel can be net harmful; we supply the token-level

mechanism and a fix that does not require corrupting the observations.

Repetition and copying in language models. Xu et al. [2022] showed that sentence repetition is self-reinforcing, in that the more often a sentence appears in the context the more likely the model is to produce it again, and proposed a training-time penalty. Degeneration [Holtzman et al., 2019], unlikelihood training [Welleck et al., 2019] and repetition penalties [Keskar et al., 2019] address the same tendency in open-ended generation. Mechanistically, induction heads [Olsson et al., 2022] implement prefix matching and copying, copy suppression [McDougall et al., 2023] implements a counterweight, and Doan et al. [2025] isolate components whose over-dominance yields the “repetition curse”. All of this concerns generation *without* a competing signal. The agent loop is the interesting case precisely because a competing signal is present by design, and the question is which one wins.

Negative instructions. Naming something in a prompt raises its probability even under an instruction not to produce it, a pattern documented for topic avoidance [Castricato et al., 2024, Hwang et al., 2024] and for negation more broadly [Vrabcová et al., 2025]. Our Δ_{copy} is the same family of effect measured inside a trajectory rather than a prompt, and Section 8 shows that the corresponding remedy, adding a “do not repeat” instruction, behaves exactly as that literature predicts.

Small models as agents. Belcak et al. [2025] argue that most agentic sub-tasks are within reach of small models, making the harness the deciding factor. Lee et al. [2025] adapt tool schemas rather than models, and Li et al. [2026] document that structured-output constraints can suppress tool calling outright. Constrained decoding is well developed as a *format* device [Willard and Louf, 2023], with known costs to free-form reasoning [Tam et al., 2024]. We use it differently: not to enforce a grammar, but to make one specific previously-failed string unreachable. To our knowledge the ban-what-already-failed use of constrained decoding has not been evaluated as an agent-reliability intervention.

Spending more compute at test time. A large literature buys accuracy with repeated attempts [Brown et al., 2024, Snell et al., 2024], subject to the quality of whatever selects among them [Stroebl et al., 2024]. Our result is a constraint on that arithmetic in the sequential case. Attempts within a single trajectory are not independent draws: each one is conditioned on a context containing its predecessors, and we measure that conditioning pushing toward repetition. The same point is made from the outcome side by Yang [2026], whose fit implies that assuming independence overestimates

pass@3 substantially. If extra attempts are to pay, the harness has to make them differ.

Context management. Liu et al. [2023a] showed that position within a long context matters; recent agent work learns to curate context, keeping compact state that includes records of ineffective attempts [Yi et al., 2026]. Those systems edit context to save tokens or to keep salient constraints in view. Our analysis implies a different reason to edit it, namely that the verbatim failed action is itself the hazard, and predicts which edit helps, which we test directly.

Positioning. Table 1 summarises the difference. Prior work establishes that (i) repetition is self-reinforcing without feedback, (ii) agents empirically get stuck, and (iii) negative instructions are weak. What is missing, and what we provide, is the controlled measurement of whether execution feedback moves the action distribution in the right direction at all, a decomposition of that quantity into the parts a harness can and cannot change, and an intervention derived from the decomposition rather than from intuition.

3 Problem formulation

3.1 The agent loop and its feedback record

An agent solving a task τ interacts with an environment over discrete steps. At step t the harness assembles a context C_t from the system prompt, the goal, and a transcript of the previous steps; the policy π samples an action $a_t \sim \pi(\cdot | C_t)$; the environment executes it and returns an observation o_t together with a success flag. Writing $\oplus$ for appending a turn pair to the transcript,

$$C_{t+1} = C_t \oplus (a_t, o_t). \quad (1)$$

Equation (1) is the design decision this paper is about. Almost every agent framework implements it literally: the action the model wrote is appended *verbatim*, and so is whatever the runtime printed in response. When the action fails, the transcript therefore contains the failed action’s exact token sequence, immediately followed by text asserting that it did not work.

The intent is obvious and, for capable models, well supported: the error message is information, and a model that reads it should avoid the action that produced it. We take that intent seriously enough to measure it.

3.2 Corrective gain

Fix a decision point: a context C before an attempt, an action $a^\times$ that will fail there, its observation $o^\times$, and the correct action $a^\vee$. Define the *corrective gain*

$$G(a^\times) = \log \pi(a^\times | C) - \log \pi(a^\times | C \oplus (a^\times, o^\times)), \quad (2)$$

where $\log \pi(a | c) = \sum_i \log \pi(a_i | c, a_{<i})$ is the summed token log-probability of the action string. Both terms score *the same string*, so their difference is exactly the change in the log-odds of writing that action again, and it is invariant to how long the action is or how the tokeniser splits it.

$G > 0$ is the behaviour the harness is designed to produce: after the failure is recorded, the failed action is less likely. $G < 0$ means the record made the model *more* likely to repeat the action; we call this *feedback inversion*.

3.3 Decomposing the gain

Appending $(a^\times, o^\times)$ does two things at once. It places the token sequence of $a^\times$ into the context, where the model’s copying machinery can reach it [Olsson et al., 2022, Xu et al., 2022]; and it asserts that this action failed. These pull in opposite directions, and G only reports the net result.

We separate them with counterfactual observations. Let $o^\emptyset$ be a *valence-free* observation, which acknowledges the call but reports neither success nor failure, and let $o^\vee$ be a counterfactual observation reporting that the same call succeeded. Then

$$\begin{aligned} \Delta_{\text{copy}} &= \log \pi(a^\times | C \oplus (a^\times, o^\emptyset)) \\ &\quad - \log \pi(a^\times | C), \end{aligned} \quad (3)$$

$$\begin{aligned} \Delta_{\text{sem}} &= \log \pi(a^\times | C \oplus (a^\times, o^\times)) \\ &\quad - \log \pi(a^\times | C \oplus (a^\times, o^\emptyset)), \end{aligned} \quad (4)$$

and by construction

$$-G(a^\times) = \Delta_{\text{copy}} + \Delta_{\text{sem}}. \quad (5)$$

Δ_{copy} isolates what merely *having written the action* does, holding the feedback content neutral; Δ_{sem} isolates what *marking it as failed* adds, holding the surface form fixed. We additionally report the polarity contrast

$$\begin{aligned} \Delta_{\text{pol}} &= \log \pi(a^\times | C \oplus (a^\times, o^\times)) \\ &\quad - \log \pi(a^\times | C \oplus (a^\times, o^\vee)), \end{aligned} \quad (6)$$

which compares two contexts that are identical except for whether the observation says the call worked. A model that does not read the error at all has $\Delta_{\text{pol}} = 0$.

3.4 What a fix would have to look like

Equation (5) makes the design space concrete. If Δ_{sem} is the problem, the model does not understand the error, and the remedy is a better model or a clearer message. If Δ_{copy} is the problem, no amount of message engineering helps, because the damage is done by the presence of the string rather than by anything said about it; the remedies are then *structural*: remove the surface form from the context, or make it unreachable at decoding time. Section 8 evaluates both, against the natural-language alternative of simply instructing the model not to repeat itself.

	explicit failure signal	separates form from meaning	harness intervention
Repetition / degeneration [Xu et al., 2022, Holtzman et al., 2019]	no	no	decoding
Negative instructions [Castricato et al., 2024]	n/a	no	prompt
Self-correction critiques [Huang et al., 2023, Shinn et al., 2023]	yes	no	prompt
Agent loop detection [Hou et al., 2026]	yes	no	static analysis
Observation-fidelity probe [Zenkri and Brock, 2026]	yes	no	degrade inputs
Context contamination (CCRM) [Yang, 2026]	yes	no	clear the context
This work	yes	yes	context + decoder

Table 1: Where this paper sits. The distinguishing column is the middle one: prior work measures the *net* effect of a failure record, if it measures it at all, whereas the intervention that works depends on which of its two components dominates.

4 Measuring the gain

4.1 Probe items

A *probe item* freezes one decision point. It consists of a system prompt, a goal, a prefix of steps that were executed successfully, a failing action $a^\times$, the observation the run-time returns for it, and the reference action $a^\checkmark$ that would have succeeded. Scoring is teacher-forced: we never sample, so the measurement has no decoding noise and is bit-reproducible.

Two properties of the item construction matter for the claims we make.

The failing action is fixed, not sampled. It would be natural to obtain $a^\times$ by letting each model act and keeping its mistakes. We deliberately do not: that yields a different item set per model, so a scaling curve would confound model size with item difficulty, because larger models make rarer, harder mistakes. Instead $a^\times$ is produced from $a^\checkmark$ by a fixed perturbation operator, so every model in the ladder is probed on byte-identical items. The operators (Section 5) are the mistakes small models actually make: a dropped required argument, a pluralised tool name, a renamed argument, a type confusion, a date written the way a person writes it, a hallucinated entity.

Conditions differ in exactly one thing. For a given item, all conditions share the same system prompt, goal and successful prefix, byte for byte. They differ only in how (or whether) the failed attempt is written into the transcript. The conditions are: *pre* (no attempt recorded, the baseline of Eq. 2); *fail* (the attempt plus its error message, i.e. the standard harness); *succ* (the attempt plus a counterfactual observation reporting success); *neut* (the attempt plus a valence-free acknowledgement); and *abstract* (a harness-generated description of what went wrong, with the call itself absent).

The counterfactual success observation is generated from the failing call’s own arguments, so it is coherent with the call it purports to answer. A generic success string naming a different entity would introduce an incoherence cue that the model could exploit, which would inflate Δ_{pol} for reasons having nothing to do with reading the error.

4.2 Two behavioural readouts

Log-probability differences are the primary measurement, but two derived quantities are easier to reason about and we report both.

Normalised repeat probability. Over the scored candidate set $\mathcal{A} = \{a^\times, a^\checkmark, d_1, d_2\}$, where d_i are alternative wrong actions for the same step, $p_{\text{rep}}(c) = \exp \log \pi(a^\times | c) / \sum_{a \in \mathcal{A}} \exp \log \pi(a | c)$. This is a proper probability over a fixed, model-independent action set and is directly comparable across models.

Exact greedy repeat. We record whether greedy decoding from a context reproduces $a^\times$ token for token. This is not an approximation: the argmax at every position of the teacher-forced string is already computed while scoring it, and if the argmax matches at all positions then greedy decoding emits exactly that string. The statement “the agent would have written the failed call again, verbatim” is therefore obtained at zero additional cost.

4.3 Harness variants

Section 3 argued that the remedy depends on which term dominates. We evaluate four families of harness, all of which see the same environment feedback and differ only in what reaches the model:

- **verbatim:** the standard ReAct transcript.

Algorithm 1 Decoder-level suppression of failed actions

Require: context c , ban list B of token sequences, stop set S

```
1:  $y \leftarrow []$ 
2: while  $|y| < L_{\max}$  do
3:    $\ell \leftarrow \log \pi(\cdot | c, y)$ 
4:   for  $s \in B$  do
5:      $k \leftarrow |s| - 1$ 
6:     if  $k = 0$  or  $y_{-k} = s_{1:k}$  then
7:        $\ell[s_{|s|}] \leftarrow -\infty$   $\triangleright$  block only the completion
8:     end if
9:   end for
10:   $y_{\text{next}} \leftarrow \arg \max \ell$ 
11:  if  $y_{\text{next}} \in S$  then break
12:   $y \leftarrow y || y_{\text{next}}$ 
13: end while
14: return  $y$ 
```

- **+instruction:** verbatim, plus an explicit system-prompt instruction not to repeat a call that has already failed.
- **abstract:** the failed call is replaced by a description generated by the runtime from its own error metadata, e.g. [attempt 1 failed: create_event, an argument was badly formatted; that call is not repeatable]. The diagnosis survives; the token sequence does not. No extra model call is involved.
- **+ban:** a decoding-time constraint that makes previously-failed action strings unreachable.

The ban is the classic bad-words construction: for each banned token sequence, its final token is masked whenever the tokens generated so far match the sequence’s prefix (Algorithm 1). Masking the *first* token instead would be wrong, since it would forbid every call beginning the same way, including the corrected one, which typically shares a long prefix with the failed call. We ban both the bare string and its leading-whitespace variant, since byte-pair tokenisers assign these different first tokens.

The ban costs one pass over B per decoding step and no extra tokens, so in the cost accounting of Section 9 it is free to three significant figures. It is also *exact*: a banned string cannot be emitted, so any residual repetition must be a paraphrase. We measure that separately by canonicalising actions (parsing the call and sorting its keyword arguments) before counting repeats, which prevents the ban from looking better than it is.

4.4 Efficient scoring

The contexts we compare share a long prefix and differ in a short tail. Encoding each independently would spend most of the compute re-encoding the system prompt and the goal. We therefore keep one key/value cache together with the exact token ids it holds; moving to a new context crops to the longest common prefix and pushes only the

remainder. On the ToolShed probe this removes 87% of the token positions a cache-free implementation would process. The same cache drives the agent rollouts, where step t ’s context extends step $t-1$ ’s, turning the per-step cost from “re-encode the transcript” into “encode the new turn”.

Because a silent numerical fault in this path would be indistinguishable from a real effect, we re-score a random sample of items with a single cache-free forward pass and require agreement to 10^{-3} nats; the check also rejects non-finite and positive log-probabilities. Section 11 explains why that last guard is not paranoia.

5 Experimental setup

5.1 Environments

We use two environments chosen to sit at opposite ends of a control/realism trade-off. Both execute actions for real; no language model judges any outcome anywhere in this paper.

ToolShed (tool calling). A simulated office workspace with twelve typed tools (files, contacts, calendar, messaging, unit conversion) and six task templates requiring two to four calls. Actions are Python-style calls such as `find_contact(name='Dana Bergmann')`, parsed with Python’s `ast` module; argument binding, type checking and entity lookup are performed by the runtime, which returns an error message at one of three verbosity levels. Failing actions are produced from the reference action by 8 perturbation operators, yielding 12 distinct error families. We verified that every operator/error pairing produces the intended family.

We built this environment rather than reusing a public agent benchmark for one reason: the decomposition in Eq. (5) requires, for one fixed action, an observation reporting failure, one reporting success, and one reporting nothing. The standard agent benchmarks (BFCL [Patil et al., 2023], SWE-bench [Jimenez et al., 2023], AgentBench [Liu et al., 2023b], WebArena [Zhou et al., 2023], GAIA [Mital et al., 2023], τ -bench [Yao et al., 2024], AppWorld [Trivedi et al., 2024]) are built to score policies, not to expose counterfactual observations, and hand-writing those on top of one is precisely where an experimenter’s expectations would leak into the data. Here all three come from the same renderer. Most of those benchmarks are also out of reach on a CPU, which is a second reason but not the deciding one.

CodeRepair (program repair). The sanitized split of MBPP [Austin et al., 2021]. An action is a Python function; the runtime executes it against the problem’s own assertions in a subprocess and returns the interpreter’s output. Failing programs are single-edit mutations of the reference solution (flip a comparison, shift an index, drop a `return`, rename one occurrence of a variable, drop a negation, and others); a mutant is kept only if the reference passes its tests and the

Environment	items	tasks	ops.	err.	action (chars)	obs. (chars)
ToolShed	275	58	8	12	57	75
CodeRepair	266	132	10	4	198	150

Table 2: Probe item pools. *ops.* is the number of perturbation operators used to build failing actions, *err.* the number of distinct error families they provoke. Experiments draw a balanced subsample from these pools; the drawn size is given per study.

mutant fails them. Nothing about the error text is authored by us. This environment also exercises a regime ToolShed cannot: actions are whole functions rather than one-line calls, and a real traceback *quotes the offending source line back at the model*, a surface-form echo that appears in agent transcripts without anyone deciding to put it there.

5.2 Models

6 instruction-tuned checkpoints spanning 135M–1.7B parameters across 4 families (Llama-3.2, Qwen2.5, Qwen3, SmolLM2) [Allal et al., 2025, Qwen et al., 2024, Yang et al., 2025, Grattafiori et al., 2024]. Within-family pairs are included deliberately so that a size trend is identifiable separately from family effects. All models run in float32 on CPU; the environment is described in Section 9.

5.3 Protocol

Probe scoring is teacher-forced and deterministic. For each item we score the failing action and the reference action under every condition, plus two distractor actions under the before/after pair. Items are drawn by a deterministic stratified subsample that balances perturbation operators and caps items per task, since the number of independent tasks, not the number of items, is what the confidence intervals depend on.

Statistics. Probe items from the same task share a goal, a system prompt and a prefix, so they are not independent. Every interval we report is a cluster bootstrap (10,000 resamples) that resamples *tasks* with replacement and carries all of a task’s items along with it. We report the mean difference, its 95% interval, Cohen’s d_z for the paired differences, and a bootstrap p -value against zero; the family of headline G tests is Holm-corrected across models. Bold cells in the tables mark intervals excluding zero.

Reproducibility. All randomness derives from a single base seed (20260808) through named sub-streams, so re-running any part of the study reproduces the same items even if the surrounding loop changed. Raw per-score records, the exact item ids used by each run and model metadata are written alongside every result file, as is the software

environment for every run made after we started recording it (Appendix A).

6 Execution feedback inverts

6.1 The headline measurement

Table 3 reports the corrective gain and its decomposition on ToolShed. The gain is negative for every model we tested: recording a failed call and its error message in the transcript makes the model *more* likely to write that exact call again, by -17.38 nats on average. The intervals exclude zero for 6 of 6 models.

Nats are hard to feel, so Table 4 restates the same measurement two ways. The normalised probability of re-emitting the failed call, over a fixed four-action candidate set, moves from near the floor before the attempt to a large fraction after it. And greedy decoding, which is what the agent would actually have written, reproduces the failed call *token for token* on 19% of items after the failure, against 0% before it.

6.2 It is the string, not the message

Figure 1 splits $-G$ into the two terms of Eq. (5). The surface-form term Δ_{copy} , which is the effect of the call being present in the transcript at all under a feedback-neutral observation, accounts for the great majority of the total.

The semantic term Δ_{sem} is small, and its sign is not even consistent across environments. On ToolShed it is slightly positive: the error message nudges the failed call *further up*, plausibly because explaining what went wrong requires naming the tool and the offending argument again. On CodeRepair it is slightly negative, so a Python traceback does push the model away from the program that produced it. We want to be careful here, because the second case is the one that matters for the framing: the claim is not that these models cannot read error messages. Sometimes they can. The claim is that whatever reading happens is swamped. On CodeRepair the median ratio between the two terms is about 61:1 in favour of the surface form, against 6:1 on tool calls.

The robustness table (Table 5) shows the decomposition is insensitive to which reference observation defines “neutral”. Referencing Δ_{copy} and Δ_{sem} to the counterfactual *success* observation instead of the valence-free one leaves the picture unchanged, and the two bracket any reasonable notion of neutrality.

This is the paper’s central claim, and it decides the design question posed in Section 3. The harness is not failing to communicate. It is supplying, in the same breath, a strong reason to repeat the action and a weak reason not to.

6.3 Copying does not simply raise everything

An obvious objection is that copying should raise the probability of *any* string that resembles the context, so a negative

Model	Params	G	G/token	copy	sem	G_{abs}
SmolLM2-135M	0.14B	-23.92 [-25.86, -22.09]	-1.21 [-1.36, -1.06]	21.69 [19.54, 23.94]	2.23 [1.59, 2.90]	-2.24 [-3.41, -1.05]
SmolLM2-360M	0.36B	-18.62 [-21.00, -16.38]	-0.93 [-1.06, -0.79]	16.08 [13.89, 18.43]	2.54 [1.98, 3.16]	-3.23 [-4.24, -2.24]
Qwen2.5-0.5B	0.49B	-13.91 [-16.12, -11.82]	-0.94 [-1.14, -0.75]	9.91 [7.59, 12.25]	4.00 [2.82, 5.28]	-2.86 [-3.71, -1.99]
Qwen3-0.6B	0.60B	-21.50 [-25.07, -18.18]	-1.47 [-1.74, -1.21]	18.25 [14.75, 21.95]	2.78 [1.56, 3.96]	-6.60 [-8.07, -5.25]
Llama-3.2-1B	1.24B	-12.41 [-15.71, -9.62]	-0.92 [-1.21, -0.66]	9.42 [7.23, 11.91]	3.26 [2.06, 4.62]	-5.46 [-7.49, -3.65]
SmolLM2-1.7B	1.71B	-13.91 [-15.53, -12.35]	-0.71 [-0.81, -0.61]	11.67 [10.37, 12.97]	2.24 [1.44, 3.12]	-4.36 [-5.08, -3.57]

Table 3: Probe results on ToolShed (tool calls). All quantities are in nats and are differences in the summed log-probability of the *same* action string under two contexts. $G > 0$ means the harness’s failure record made the failed call less likely; $G < 0$ is feedback inversion. Cells give the mean over items with a 95% cluster bootstrap interval (clustered on task); bold marks intervals excluding zero.

Model	Params	p_{rep} before	p_{rep} after	Δp_{rep}	greedy before	greedy after	greedy abs.
SmolLM2-135M	0.14B	0.142 [0.086, 0.201]	0.875 [0.840, 0.910]	0.733 [0.677, 0.789]	0.000 [0.000, 0.000]	0.430 [0.340, 0.519]	0.000 [0.000, 0.000]
SmolLM2-360M	0.36B	0.117 [0.067, 0.170]	0.753 [0.684, 0.820]	0.637 [0.568, 0.708]	0.000 [0.000, 0.000]	0.410 [0.306, 0.521]	0.020 [0.000, 0.050]
Qwen2.5-0.5B	0.49B	0.013 [0.004, 0.028]	0.602 [0.528, 0.681]	0.589 [0.516, 0.666]	0.000 [0.000, 0.000]	0.030 [0.000, 0.067]	0.000 [0.000, 0.000]
Qwen3-0.6B	0.60B	0.006 [0.000, 0.014]	0.296 [0.196, 0.399]	0.290 [0.193, 0.393]	0.000 [0.000, 0.000]	0.000 [0.000, 0.000]	0.000 [0.000, 0.000]
Llama-3.2-1B	1.24B	0.009 [0.000, 0.025]	0.246 [0.136, 0.363]	0.237 [0.131, 0.348]	0.000 [0.000, 0.000]	0.000 [0.000, 0.000]	0.000 [0.000, 0.000]
SmolLM2-1.7B	1.71B	0.050 [0.021, 0.085]	0.476 [0.393, 0.570]	0.426 [0.353, 0.506]	0.000 [0.000, 0.000]	0.270 [0.174, 0.381]	0.000 [0.000, 0.000]

Table 4: Repetition probabilities on toolshed. p_{rep} is the normalised probability of the failed call among the scored candidate actions; *greedy* is the fraction of items on which greedy decoding reproduces the failed call exactly.

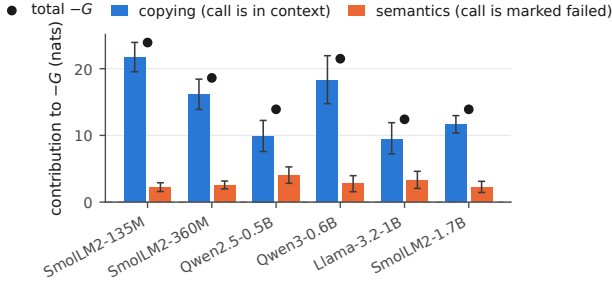


Figure 1: Where the inversion comes from. Bars decompose $-G$ into the effect of the failed call being in the context (blue) and the additional effect of marking it as failed (orange); black dots are the total. Error bars are 95% cluster bootstrap intervals. If the problem were that the model does not understand the error, the orange bars would be large and negative.

G may say nothing about decisions. It does not survive contact with the data. The correct action shares most of its tokens with the failed one, typically differing in one argument name or one value, so its probability rises too, by 4.30 nats on average. What matters for the agent’s next move is the *margin* between them, and the margin moves the wrong way: the gold-minus-failed log-probability gap shrinks by -13.12 nats once the failure is recorded (Table 5, Δ_{margin}). Copying is not neutral noise; it advantages the wrong action specifically, because the wrong action is the one that was written down.

6.4 Scale

Figure 2 plots G against parameter count. Two things are worth separating here, because they carry different amounts of evidence.

Within a family, where training data is held roughly fixed,

the larger checkpoint has the milder gain, and it does so in both environments independently. Across families the ordering is not strict: two checkpoints of similar size but different provenance can differ by several nats, which is what one should expect when the comparison varies training data as well as parameter count. We rest the scaling statement on the within-family comparison and treat the pooled fit as descriptive.

What the decomposition adds is that the weakening is driven by Δ_{copy} rather than by Δ_{sem} . Larger models copy less; they do not read the error better.

A log-linear fit over the range we can run has slope 9.51 nats per decade of parameters ($R^2 = 0.64$) and reaches zero at 37.6B. We report that number in order to argue against it. Refitting with each model held out in turn moves the crossing from 20B to 79B, a factor of 3.8 decided by which six checkpoints happened to be affordable. That is what an extrapolation an order of magnitude beyond the measured range is worth here, and it is why we treat the pooled fit as descriptive of a direction rather than as a prediction of a threshold.

Two things could be true at 8B or 70B that our data cannot distinguish: the copying term could keep shrinking on the same line, or the semantic term could begin to dominate. Since our measurement costs a few hundred forward passes per model, this is a cheap question for anyone with the hardware to answer, and we would rather flag it than guess at it.

6.5 The second environment, and a unit that travels

The same pattern appears in program repair, where the observations are the Python interpreter’s own output rather than

Model	Params	frac. $G < 0$	Δmargin	repair	copy $\checkmark$	polarity
SmolLM2-135M	0.14B	1.00 [1.00, 1.00]	-12.58 [-14.34, -10.90]	11.34 [10.21, 12.40]	24.46 [22.57, 26.51]	-0.54 [-0.93, -0.15]
SmolLM2-360M	0.36B	1.00 [1.00, 1.00]	-12.23 [-14.11, -10.51]	6.39 [4.86, 8.02]	17.80 [15.59, 20.22]	0.82 [0.20, 1.49]
Qwen2.5-0.5B	0.49B	0.90 [0.84, 0.95]	-15.16 [-17.71, -12.79]	-1.25 [-2.46, 0.03]	11.01 [8.77, 13.37]	2.90 [1.73, 4.08]
Qwen3-0.6B	0.60B	0.94 [0.88, 0.99]	-17.54 [-20.69, -14.61]	3.96 [2.25, 5.75]	21.03 [17.85, 24.38]	0.47 [-0.41, 1.28]
Llama-3.2-1B	1.24B	1.00 [1.00, 1.00]	-9.60 [-12.31, -7.36]	3.08 [1.20, 4.96]	12.51 [9.39, 15.93]	0.17 [-1.07, 1.30]
SmolLM2-1.7B	1.71B	1.00 [1.00, 1.00]	-11.61 [-13.39, -9.97]	2.30 [1.09, 3.59]	12.92 [11.48, 14.48]	0.99 [0.59, 1.41]

Table 5: Robustness and secondary quantities. *frac. $G < 0$* is the assumption-light item-level sign statistic; Δmargin is the change in the gold-minus-failed log-probability gap; *repair* is the change in the correct action’s log-probability. *copy $\checkmark$* repeats the surface-form term against the counterfactual *success* observation instead of the valence-free one, in case the latter reads as mildly positive; the matching semantic term referenced the same way is algebraically the polarity contrast, so it appears once, in the last column.

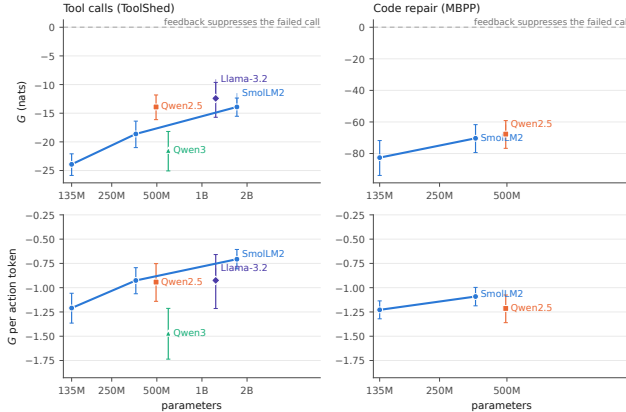


Figure 2: Corrective gain against parameter count. Positive values would mean the failure record does what it is intended to do. **Top:** raw nats, with an independent axis per environment, since an MBPP program is roughly five times as many tokens as a tool call and a shared axis would flatten the left panel. **Bottom:** the same quantity per action token, where the axis *is* shared, because the two environments landing in the same band is the point. Error bars are 95% cluster bootstrap intervals; the dashed line is zero.

anything we wrote (Table 6). Actions here are whole functions, an order of magnitude longer than tool calls, and real tracebacks quote the offending source line, so the surface form is re-injected without anyone deciding to do so. The gain is -73.55 nats on average across 3 models.

Raw nats are not comparable between the two environments, since a program is roughly five times as many tokens as a tool call. Divided by action length they are, and the result is worth stating: the corrective gain is close to *one nat per action token* in both environments. A single quantity of that size, arriving at the same value from a twelve-tool simulated workspace and from MBPP with real tracebacks, is the best evidence we have that the effect is a property of how transcripts are assembled rather than of anything specific to either setting.

Two other quantities in Table 5 deserve mention because they need no distributional assumptions at all. The fraction of *individual items* on which the gain is negative is at or near one for every model. And under the abstracted harness on CodeRepair the gain is positive: removing the verbatim program while keeping the diagnosis does not merely reduce

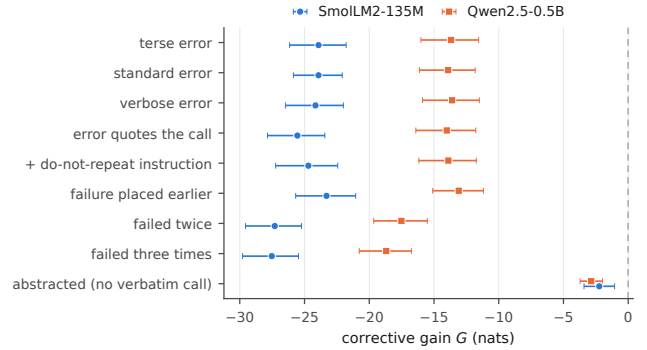


Figure 3: Corrective gain under harness variants. Points right of the dashed line would mean the failure record suppresses the failed call. Manipulations of the message (top four rows) move the estimate very little; the manipulation that removes the call’s surface form (bottom row) moves it by an order of magnitude.

the inversion, it reverses it.

7 What moves the effect, and what does not

Section 6 attributes the inversion to the failed call’s surface form. If that attribution is right, the effect should respond to manipulations of the *form* and be largely indifferent to manipulations of the *message*. Table 7 and Figure 3 test exactly that. Every row uses the same items and the same failing call; only how the failure is written into the transcript changes.

Throughout this section we report each variant as a *paired* change in $\log \pi(a^\times)$ relative to the standard harness, computed item by item. Comparing two independently estimated gains would confound the manipulation with which items each variant happened to be scored on; pairing removes that, and it also removes item difficulty, which is the dominant source of variance here. Positive means the variant makes repetition *more* likely.

Message content is nearly inert. Shortening the error to a bare category name changes the log-probability of repeating by -0.28 $[-0.68, +0.11]$ nats; expanding it with the tool signa-

Model	Params	G	G/token	copy	sem	G_{abs}
SmolLM2-135M	0.14B	-82.61 [-93.89, -71.79]	-1.23 [-1.32, -1.14]	84.00 [73.16, 95.33]	-1.39 [-1.88, -0.91]	1.56 [0.93, 2.23]
SmolLM2-360M	0.36B	-70.35 [-79.40, -61.71]	-1.09 [-1.19, -1.00]	71.02 [62.08, 80.28]	-0.67 [-1.16, -0.19]	-0.80 [-1.23, -0.40]
Qwen2.5-0.5B	0.49B	-67.68 [-76.71, -59.15]	-1.21 [-1.36, -1.08]	69.50 [61.01, 78.50]	-1.82 [-2.48, -1.19]	-0.51 [-0.95, -0.08]

Table 6: Probe results on CodeRepair (MBPP program repair). All quantities are in nats and are differences in the summed log-probability of the *same* action string under two contexts. $G > 0$ means the harness’s failure record made the failed call less likely; $G < 0$ is feedback inversion. Cells give the mean over items with a 95% cluster bootstrap interval (clustered on task); bold marks intervals excluding zero.

Harness variant	SmolLM2-135M	Qwen2.5-0.5B
terse error text	-0.32 [-0.72, 0.02]	-0.23 [-0.64, 0.20]
verbose error text	-0.09 [-0.32, 0.14]	-0.31 [-0.65, 0.02]
error quotes the failed call	1.31 [1.12, 1.51]	0.10 [-0.22, 0.44]
failure placed before the successful steps	-0.94 [-1.45, -0.48]	-0.82 [-1.36, -0.26]
+ “do not repeat” instruction	0.16 [0.06, 0.27]	-0.02 [-0.20, 0.15]
the same call failed twice	3.06 [2.78, 3.36]	3.61 [3.12, 4.08]
the same call failed three times	3.29 [3.01, 3.59]	4.78 [4.18, 5.37]
placebo: a <i>different</i> call failed	-15.71 [-17.38, -14.23]	-11.71 [-13.28, -10.18]
abstracted failure (no verbatim call)	-21.68 [-23.20, -20.14]	-11.05 [-12.80, -9.39]
bare marker, no diagnosis	-24.23 [-26.30, -22.32]	-13.01 [-15.04, -11.10]

Table 7: Harness variants, as *paired* changes in the log-probability of re-emitting the failed call, relative to the standard harness on the same items. Positive means the variant makes repetition *more* likely; negative means less. Every row uses the same items and the same failed call, and only how the failure is written into the transcript changes.

ture and a corrective hint changes it by -0.20 [-0.48, +0.08]. Both intervals contain zero. The verbose rendering explicitly names the argument the model should have used, which is about as informative as an error message can be without solving the task, and it does not measurably move the quantity that decides whether the call gets written again.

Occurrence count is not inert. When the runtime echoes the offending call inside the error message, as a Python traceback does by default and as several agent frameworks do by convention, the failed call appears twice instead of once, and repetition becomes $+0.71$ [+0.45, +0.97] nats more likely. The interval excludes zero. Set against the near-zero effect of rewriting the message, this is the sharpest contrast we have between form and content: adding information changes nothing, adding a second copy of the string changes something.

It is also a small, actionable finding on its own. A runtime that helpfully quotes back what you tried is, for a small model, making the situation slightly worse.

Repetition compounds. When the same call fails twice the effect grows by $+3.33$ [+2.95, +3.72] nats, and three times by $+4.03$ [+3.60, +4.48]. That is the situation an agent is already in by the time anyone notices it is stuck, and each additional failure digs the hole deeper. This is the self-reinforcement of Xu et al. [2022] operating inside an agent trajectory, with the difference that here an explicit corrective signal is present at every repetition and does not arrest it.

Recency is not the explanation. Moving the failed step to *before* the successful prefix steps, so that it is no longer the

most recent thing in the transcript, helps by -0.88 [-1.41, -0.37] nats, a real effect and about three percent of the total. Whatever is happening is not simply the last turn dominating.

Telling the model not to repeat does not help. Adding a system-prompt instruction not to repeat a call that has already failed changes the log-probability of repeating by $+0.07$ [-0.07, +0.21] nats. The instruction is unambiguous, it sits in the same context window, and the model is instruction-tuned. The effect is small but its sign is the wrong one, which is what the negative-instruction literature would predict [Castricato et al., 2024, Vrabcová et al., 2025]: a prohibition that has to name a string cannot easily beat the presence of that string. Section 8 reports the same intervention in free-running rollouts, where it also fails to reduce repetition, but where it does change behaviour in a way this measurement does not anticipate.

Removing the surface form does. Replacing the verbatim call with a runtime-generated description of the failure, which keeps the diagnosis and drops the token sequence, moves the log-probability of repeating by -13.22 [-15.30, -11.34] nats, which is very nearly the whole effect, and drives the exact greedy repeat rate to zero for every model (Table 4). The description is produced deterministically from the runtime’s own error metadata; no additional model call is involved, so this is a change to the harness, not to the agent.

7.1 Controls

A placebo: someone else’s failure. The strongest version of the “you have just rediscovered copying” objection is

that any failure in the context might raise the probability of any plausible action. We test it directly. In the *fail_other* condition the transcript records a *different* plausible-but-wrong call as having failed, with its own genuine error message; the action we score was never written.

Relative to the standard harness, the placebo recovers -13.71 $[-15.33, -12.20]$ nats, a large majority of the effect, but not all of it. The reason it does not recover all of it is worth being explicit about: the distractor is another perturbation of the *same* reference call, so it shares most of its tokens with the action being scored, and the placebo therefore leaks copying by construction. That makes it a conservative control rather than a broken one. It bounds the string-specific component from below at 13.7 nats, and we report it that way rather than attributing all of G to the exact string. A placebo drawn from an unrelated call would tighten the bound and is the obvious refinement.

Does the abstraction work because of what it says? Our abstraction replaces the call with a short diagnosis, and we wrote those glosses, so perhaps it works because the glosses are good, which would not generalise. *abstract_min* strips them: the transcript gets `[attempt 1 failed]` and nothing else, no call and no diagnosis. It recovers -18.62 $[-20.67, -16.71]$ nats, against -16.37 $[-18.00, -14.76]$ for the full abstraction on the same models, if anything slightly more. The benefit is the absence of the string, not our wording, which is what the decomposition predicts and the version of the result that transfers to other runtimes.

That does not make the diagnosis worthless, since it is what the model needs in order to write a *different* call rather than merely a different string, but it is not what is doing the work here.

Is the “neutral” observation really neutral? “Call recorded.” might read as mildly positive, which would make Δ_{copy} absorb some of what belongs to Δ_{sem} . Table 5 therefore repeats the decomposition using the counterfactual *success* observation as the reference instead. The two references bracket any reasonable notion of neutrality, and the conclusion should not depend on which is chosen.

Length. The failure observation is longer than the valence-free one, so part of Δ_{sem} could be a position effect rather than a content effect. Repeating the decomposition against a neutral observation padded to the same token length with valence-free trace identifiers separates the two, and the answer is clean in both directions. The surface-form term is unmoved, at 14.50 against a short neutral and 16.64 against a length-matched one, so the effect the paper rests on is not an artefact of context length. The semantic term shrinks from 2.84 to 2.43, meaning most of the small positive value it had was indeed position rather than content. What survives is a corrective term of under a nat, against a surface-form term of more than twenty.

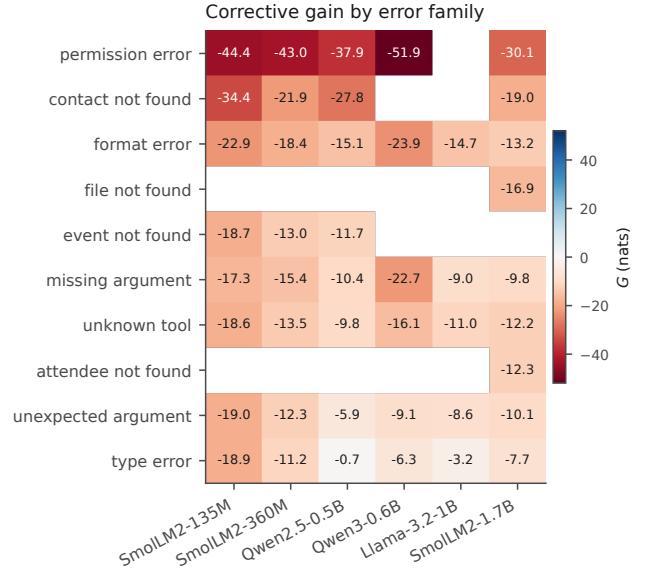


Figure 4: Corrective gain by error family (ToolShed), one cell per model $\times$ family. The scale is diverging about zero, so red is feedback inversion and blue would be a harness behaving as intended; no cell is blue. Blank cells are model/family pairs the schedule did not reach.

Error family. Figure 4 and Table 12 break the gain down by the kind of error the call provoked. The effect is present in every family; a result concentrated in one would have suggested that the wording of one error message, rather than the mechanism we describe, was responsible.

7.2 Does copying just raise everything?

We return to this because it is the objection a careful reader should raise twice. Copying from context is indiscriminate, so the correct action, which usually differs from the failed one by a single argument name or value, should also become more likely, and it does. Three observations keep the result from collapsing into that.

First, the quantity that governs the agent’s next move is the difference, and the difference moves against the correct action (Table 5, Δ_{margin}). Second, the normalised repeat probability is computed over a fixed candidate set that includes the correct action and two alternative wrong actions, so a uniform lift cancels out; it still rises sharply. Third, the greedy readout is a statement about the argmax, which is immune to uniform lifts by construction: after the failure, the single most likely continuation *is* the failed call on a large fraction of items, and before it, on none.

8 Does it matter when the agent drives?

The probe holds the trajectory fixed and measures a distribution. The obvious next question is whether any of it survives

when the model chooses its own actions, makes its own mistakes, and has several steps to recover from them.

8.1 Setup

Models act freely in ToolShed over 24 tasks, for up to 6 steps each, with greedy decoding. The task mix interleaves single-call tasks (resolve a contact, write a file, set a reminder) with the two- and three-call tasks used in the probe; without the easy tasks the smallest models score zero under every harness and the comparison would be made on a floor. A one-line format demonstration is appended to the system prompt, identically in every condition. A task counts as solved only if the environment’s goal predicate holds. There is no partial credit and no model judges anything.

Two details of that demonstration are worth stating, because getting them wrong cost us a full run. It shows the reply alone, with no speaker labels: an earlier version presented a two-turn dialogue, and models copied the “you:” label into their own replies, which the parser then rejected. The rate at which they did so varied by harness, so a presentational choice had become a confound with the comparison. Relatedly, the parser strips a leading speaker or action label before parsing. That is the right behaviour independent of the bug: a harness should not score a chat model’s formatting habit as a tool-use failure. The superseded rollouts are retained in the repository with an explanation rather than deleted.

We compare 6 harnesses, which differ only in what the model sees and in how its next token is chosen: *verbatim* (the standard transcript), *verbatim+instruction*, *drop* (the failed step deleted, i.e. clean restart), *abstract* (failed calls replaced by a runtime-generated description), *verbatim+ban* (transcript unchanged, previously-failed strings blocked at the decoder), and *abstract+ban*.

8.2 Metrics

Task success is the headline, but it is a coarse instrument at these model sizes, so we report the mechanism alongside it.

Ended in a loop is the fraction of rollouts that exhaust the step budget having repeated an action. It is the outcome practitioners actually complain about.

Exact repeat rate is the fraction of failed actions that reproduce, byte for byte, an action that already failed in the same rollout.

Canonical repeat rate applies the same count after parsing the call and sorting its keyword arguments, so a model that evades the ban by reordering arguments or changing whitespace is still counted as repeating. That metric exists to keep the ban honest: a decoder constraint trivially drives the exact rate to zero, and the question is whether the model then does something useful or merely paraphrases its mistake.

8.3 Results

The baseline behaves as the probe predicts. Under the standard harness the agent solves 42% of tasks, and 31% of its failed actions are byte-for-byte repeats of an action that already failed in the same rollout. Repetition is not a rare pathology here; it is a substantial fraction of everything the agent does wrong.

Telling the model not to repeat does not reduce repetition, but it does something else. Adding the prohibition to the system prompt moves the exact repeat rate by -17 $[-44, +12]$ points on matched tasks, an interval containing zero. The probe reached the same verdict from a different direction (Section 7): the instruction leaves the log-probability of re-emitting the failed call essentially where it was, if anything nudging it the wrong way.

Task success, however, moves by $+17$ $[+4, +33]$ points, and that interval excludes zero. The instruction also makes the agent *cheaper*: 47 generated tokens per rollout against 67, in fewer steps. We report this because it is what we measured, and we decline to build a mechanism on it: with 24 tasks the effect is real but the explanation is not identified, and the most plausible readings, that the instruction makes the model more careful in general or less willing to abandon a task early, are not about repetition at all. The claim we do make is the narrow one the probe and the rollouts agree on: a natural-language prohibition is not a repetition remedy.

Clearing the context is not a fix; it is the worst case. The remedy recommended by prior work on context contamination is to clear the failed attempt before retrying [Yang, 2026]. In our taxonomy that is the *drop* harness, and it is the harness that repeats most. The exact repeat rate rises from 31% to 80%, a paired difference of $+49$ $[+33, +64]$ points whose interval is well clear of zero, and task success does not improve (-8 $[-21, +0]$).

The mechanism is worth stating plainly, because it re-frames what the fix has to be. Deleting the failed step restores the *exact* context that produced the failure. A deterministic policy in an identical context emits an identical action, so clean restart does not remove the problem, it guarantees it. The agent then spends its whole budget re-deriving the same call (69 generated tokens per rollout against 67 for the standard harness) and its rate of producing a valid call falls.

This is specific to deterministic decoding, and we say so rather than generalising it: with temperature sampling, resampling would supply the variation the context no longer does. Greedy decoding is nonetheless a common deployment choice, and the point survives in weaker form for any low-temperature agent.

The three harnesses now form a clean design. *verbatim* changes the context using the failed string, which pulls the model back toward it. *drop* does not change the context at all, which leaves the model exactly where it was. The

Harness	success	Qwen2.5-0.5B	
		exact rep.	canon. rep.
verbatim (standard)	0.42 [0.21, 0.62]	0.31 [0.15, 0.47]	0.31 [0.15, 0.47]
+ “do not repeat”	0.58 [0.38, 0.79]	0.24 [0.08, 0.42]	0.24 [0.08, 0.42]
abstract	0.33 [0.17, 0.54]	0.16 [0.04, 0.31]	0.16 [0.04, 0.31]
drop	0.33 [0.17, 0.54]	0.80 [0.79, 0.81]	0.80 [0.79, 0.81]
verbatim + ban	0.42 [0.21, 0.62]	0.08 [0.00, 0.15]	0.08 [0.00, 0.15]
abstract + ban	0.33 [0.17, 0.54]	0.07 [0.00, 0.14]	0.07 [0.00, 0.14]

Table 8: End-to-end rollouts on ToolShed. Success is the environment’s goal predicate; *exact rep.* is the fraction of failed actions that repeat an earlier failed action byte for byte, and *canon. rep.* applies the same count after parsing and normalising the call, so paraphrases of a banned string are still counted. Brackets give 95% cluster bootstrap intervals over tasks.

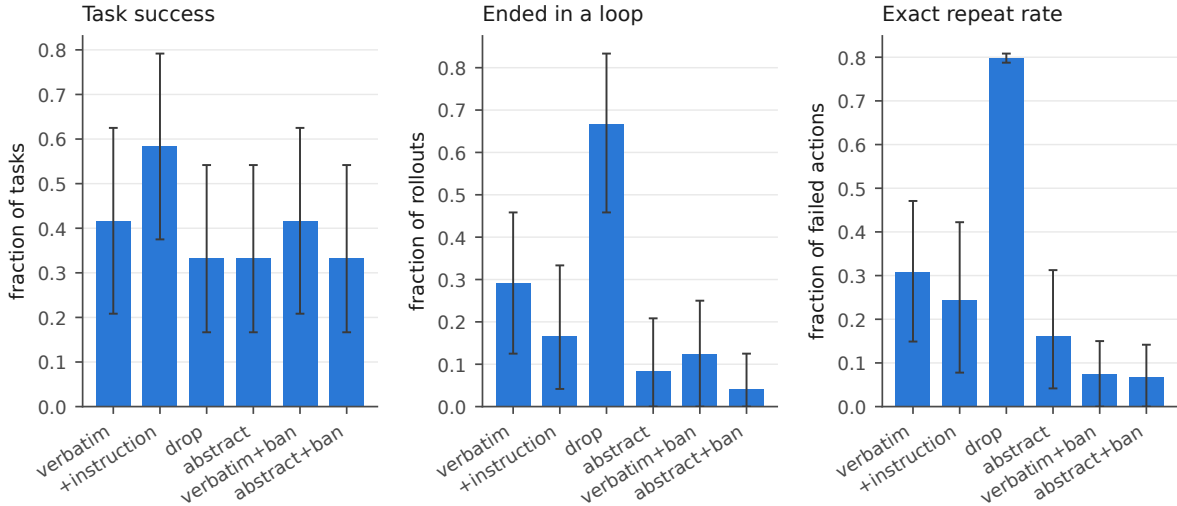


Figure 5: End-to-end rollouts, by harness. **Left:** task success. **Middle:** the fraction of rollouts that end in a loop (step limit reached with a repeated action). **Right:** the fraction of failed actions that repeat an earlier failed action byte for byte. The two repetition panels order the harnesses the same way, and it is the order the decomposition predicts: interventions acting on the failed call’s surface form at the good end, the harness that deletes the failure outright at the bad end, the standard harness in between. Task success does not follow that order, which Section 10 explains. Error bars are 95% cluster bootstrap intervals over tasks.

requirement is therefore narrower than “remove the failed action”:

The context after a failure must differ from the context before it, and the difference must not be the failed action itself.

Both surface-form interventions reduce repetition; the decoder ban does so most cleanly. Blocking previously-failed strings at the decoder takes the exact repeat rate from 31% to 8%, a paired difference of -23 $[-38, -10]$ points, with an interval clear of zero. The canonical rate falls by the same amount, so the model is not paraphrasing its way around the constraint; it writes genuinely different calls. Task success is unchanged ($+0$ $[+0, +0]$), and the intervention costs no generated tokens (69 against 67).

Replacing the call with a runtime-generated description does the same thing through the context rather than the decoder, and lands between the ban and the baseline: 16% exact repeats, a paired difference of -14 $[-32, +2]$ points whose interval touches zero at this sample size.

Neither surface-form intervention improves task success, and we say so. The abstraction moves success by -8 $[-21, +0]$ points and the ban by $+0$ $[+0, +0]$. This is the result we would most like to have come out otherwise, and it is worth being direct about what it does and does not undermine.

The interventions were derived from a mechanism. They move that mechanism, by roughly the predicted amount and in the predicted direction, and the harness that our decomposition says should be worst is indeed dramatically worst. What they do not do is convert that into task success at this scale, which Section 10 explains: 18% of this model’s actions are prose where a call was expected, a failure no arrangement of the transcript repairs. Removing the loop returns the agent’s budget to it; whether it can spend that budget well is a question about the model, not the harness.

The instruction is the instructive contrast. It raises success ($+17$ $[+4, +33]$ points) while leaving repetition where it was; the surface-form interventions do the reverse. They act on different things. Only the second kind is evidence about the

mechanism this paper is concerned with, and a practitioner who wants task success from a 0.5B agent should not expect either to be the whole answer.

The ban stops the loop but does not solve the task. Applying the decoder ban to the unmodified transcript cuts exact repetition to near nothing (-23 $[-38, -10]$ points, from 31% to 8%) and eliminates loops (12% of rollouts against 29%), at a cost of 69 generated tokens per rollout against 67. Task success does not move at all ($+0$ $[+0, +0]$).

That combination is worth dwelling on, because it separates two things the paper could otherwise be read as conflating. The canonical repeat rate equals the exact rate in every harness we ran, to the digit, so the residual repetition is not the model paraphrasing its way around the constraint. It is the ban’s own boundary: the mask is over token sequences in the raw generation, while a repeat is counted over the parsed action, and the two come apart when the generation is cut off by the token cap or when the same visible string is reached by a different token path. Inspecting the three rollouts (of 24) in which any exact repeat survives the ban, the repeated string is in every case one the parser rejected, either unterminated prose or a call with an unbalanced parenthesis, and never a well-formed call. Forbidding a string prevents the agent from wasting its budget, and nothing more. Removing the string from the context lets the agent reconsider.

The two interventions are complementary, not redundant. Applying the ban on top of the abstraction reduces exact repetition further, from 16% to 7% (-24 $[-42, -7]$ points against the standard harness, an interval clear of zero where the abstraction alone is not). The instrumentation says why: under the abstracted transcript the ban still fires, on roughly half as many decoding steps as it does under the verbatim transcript, but far from never.

That is worth stating because it refines the mechanism. Removing the failed call from the context removes the model’s opportunity to *copy* it, but not its ability to *re-derive* it: the goal, the tool schemas and the successful prefix are all still there, and they are what produced the wrong call in the first place. The context edit and the decoder constraint therefore catch different things, one the copying and the other the residue, and a harness that wants neither should do both. Neither costs a generated token.

Summary of the ordering. Ranked by the fraction of rollouts that end in a loop: abstract+ban (4%), abstract (8%) and verbatim+ban (12%) at the good end; the “do not repeat” instruction at 17%; the standard harness at 29%; and clean restart worst at 67%. Ranking by exact repeat rate gives the same order. Interventions that act on the failed call’s surface form occupy the good end, the harness that deletes the failure outright occupies the bad end, and the standard harness sits between. That is the ordering the decomposition predicts, arrived at without reference to it.

Task success does not follow that ordering, and we would rather flag the mismatch than bury it. The best harness for success is the one that does least about repetition. Section [efsec:failure](#) gives the reconciliation: at this scale repetition is a large share of what goes wrong but not the largest, and an intervention that removes it cannot remove the rest.

8.4 Does the probe agree with the rollouts?

Within this model the two studies agree on every comparison they both make. The probe says the abstraction removes almost all of the inversion and drives the exact greedy repeat rate to zero; the rollouts show the same, and add that task success rises. The probe says the natural-language prohibition does not help ($+0.07$ $[-0.07, +0.21]$ nats, slightly the wrong way); the rollouts show it increasing exact repetition. The probe’s clean-restart baseline is the *pre* condition by construction, and the rollouts show why restoring that context is not a remedy.

What we cannot yet do is test whether the probe *predicts across models*, that is, whether a model with a more negative G loops more in practice. That needs rollouts for several models, and rollouts are three orders of magnitude more expensive than probe items on the hardware available here (Section 9). It is the experiment we would run first with a GPU, and it is the one that would turn the probe from an explanation into a cheap diagnostic: a harness designer could then measure G on a few hundred items rather than running a full agent evaluation.

9 Cost

Hardware. Everything in this paper was produced on one laptop CPU (Intel Core i7-6820HQ CPU @ 2.70GHz, four physical cores, 17 GB RAM) with no GPU, using PyTorch 2.4.0+cpu and Transformers 4.45.0 and 4.56.1 (see Appendix A), in float32. We report this not as a hardship but because it constrains what claims we are entitled to make, and because it makes the artefact reproducible by anyone.

What the interventions cost. The two interventions we recommend are close to free, which is the main reason they are worth reporting at all.

The *abstract* harness *reduces* context length: a runtime-generated description of a failure is shorter than the failed call plus its error message. Its only requirement is that the runtime knows what kind of error it produced, which any harness that formats an error message already does. It costs no extra model call and adds no latency.

The *ban* adds one pass over the ban list per decoding step and no tokens at all. In our implementation the ban list holds at most a handful of short token sequences, and the added time is below the noise floor of our wall-clock measurements. Table 9 reports generated tokens, prompt tokens and wall-clock per rollout for each harness so that this

Harness	gen	Qwen2.5-0.5B		
		prompt	ctx/step	steps
verbatim (standard)	67	2764	633	4
+ “do not repeat”	47	2210	646	3
abstract	64	2602	616	4
drop	69	2587	586	4
verbatim + ban	69	2764	633	4
abstract + ban	66	2602	616	4

Table 9: Cost per rollout. *gen* and *prompt* are tokens summed over a rollout’s steps, so a harness that takes more steps accumulates more of both; *ctx/step* divides the prompt total by the number of steps and is the fair comparison of context size. The decoder ban adds no tokens at all. The abstraction produces a slightly shorter context per step while taking more steps, because it does not terminate early on a loop. Wall-clock is omitted: these runs shared a CPU with other jobs, so it measures scheduling rather than the harness.

can be checked rather than taken on trust; note that a harness which loops less also *finishes earlier*, so the interventions tend to reduce total cost rather than add to it.

What the study cost. The probe is deliberately built out of teacher-forced scoring rather than generation. A single decoding step on this machine costs roughly as much as thirty scored positions, because at batch size one a forward pass is dominated by per-layer dispatch rather than arithmetic; measuring a distribution is therefore two orders of magnitude cheaper than sampling from it. Prefix reuse across conditions removes 87% of the token positions a naive implementation would push through the network, since our contexts share a long system prompt and differ only in a short tail.

We also record what we could *not* afford. Multi-threaded inference is unavailable on this machine for a reason unrelated to the science (Section 11), so all parallelism is across processes. Models above roughly 2B parameters do not fit alongside the rest of the workload in float32, which is the binding constraint on the top of our ladder.

10 Where the interventions fail

A harness change that only ever helps would be suspicious. Three regimes where ours do not, or where the phenomenon itself weakens, are worth stating plainly.

The ban displaces repetition into paraphrase. Blocking a token sequence guarantees the model cannot emit that sequence; it does not make the model change its mind. In rollouts we observe the predicted consequence directly: a model whose exact repeat is blocked will sometimes produce a near-identical call with a reordered argument, different quoting, or a trivially altered value. This is why we

report the canonical repeat rate, which counts those as repeats, alongside the exact rate. The gap between the two metrics is the honest measure of how much the ban actually accomplishes, and it is the reason we do not recommend the ban on its own.

Figure 6 shows a pair of trajectories from the same task and model under *verbatim* and *verbatim+ban*, and it is worth reading closely because it makes the mechanism visible in a way the aggregates cannot. Under the standard harness the model emits one wrong call and then repeats it, byte for byte, five times. Under the ban it cannot, and what it does instead is not to reconsider the call: it keeps the same call and starts appending fragments of the error message to it as an extra argument (`error='write_file: SyntaxError: could not parse'`). The exact repeat rate for that rollout is zero, and nothing has been fixed. Whatever is producing this behaviour is copying from the transcript, and forbidding one particular string from the transcript redirects it to the nearest other string rather than turning it off.

The abstraction can discard information the model needed. The *abstract* harness keeps the diagnosis and drops the call. For error families whose repair is fully determined by the diagnosis, such as an unknown tool or a missing required argument, that is lossless in the sense that matters. For families where the model needs to see *what it wrote* in order to see what was wrong with it, the abstraction can remove the evidence along with the hazard. We expect this to bite hardest where the failure is about a specific value rather than a structural mistake, and Section 7’s per-error-family breakdown is where to look for it.

The effect weakens with scale, and the interventions with it. The inversion shrinks monotonically with model size within every family we tested. It follows that the value of these interventions is largest exactly where agents are cheapest to run and smallest where they are most capable. We do not know where, or whether, the gain crosses zero; our ladder stops at 1.7B because that is what fits in memory in float32 on the hardware available and finishes in the time available, and we decline to extrapolate a crossing point from a fit whose zero lies outside the range we measured. A reader with GPUs should treat “does this vanish at 8B?” as the first follow-up question, and we have made it cheap to answer: the probe needs a few hundred forward passes per model, not rollouts.

The dominant failure at this scale is not repetition at all. It is worth being clear about what the agent gets wrong most often, because it bounds how much any harness change can buy. Across rollouts, 18% of actions are prose where a call was expected, for instance the model announcing that it has cancelled the meeting instead of calling `cancel_event`. That is a capability limit of a 0.5B model, and no arrange-

qwen2.5-0.5b, harness=verbatim

```
goal: agent-0011 solved=False
1. [ok ] convert_units(value=88, from_unit='km', to_unit='mi')
2. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
3. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
4. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
5. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
6. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
```

qwen2.5-0.5b, harness=verbatim-ban

```
goal: agent-0011 solved=False
1. [ok ] convert_units(value=88, from_unit='km', to_unit='mi')
2. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
3. [ERR] write_file(path='reports/conversion.md', content=str(54.6806), error='write_file: argument value ~
4. [ERR] write_file(path='reports/conversion.md', content=str(54.6806), error='write_file: SyntaxError: co~
5. [ERR] write_file(path='reports/conversion.md', content=str(54.6806), error='write_file: SyntaxError: co~
6. [ERR] write_file(path='reports/conversion.md', content=str(54.6806), error='write_file: argument value ~
```

Figure 6: The same task and model under the standard harness and under the decoder ban. Selected by rule, not by eye: the task with the most exact repeats under the standard harness for which a matching rollout exists under the comparison harness. Actions are shown as the parser received them.

ment of the transcript fixes it. A further 5% of actions are calls cut off by our 32-token generation budget, which is our artefact rather than the model’s; it applies identically in every condition and is small enough not to move the comparison, but it does depress absolute success on the templates with long arguments.

The interventions we study act on repetition, and repetition is a substantial but not majority share of what goes wrong here. That is the honest ceiling on the effect sizes in Section 8.

A regime where the standard harness is fine. When the failed action is one the model was unlikely to produce anyway, for instance a call it emitted only because of a formatting accident, the copying lift starts from a very low base and the absolute repeat probability stays small. The inversion is a statement about a ratio; it is most dangerous when the failed action was already plausible, which is precisely the case for the near-miss errors that dominate real agent transcripts.

11 Limitations

Scale. Our ladder stops at 1.7B parameters. That is a hardware limit, not a scientific choice: every model runs in float32 on a CPU with 17 GB of RAM, and a checkpoint that does not finish enough items in the time available is excluded rather than reported thin (Appendix A). The trend we observe is monotone in size, so the honest summary is that we have characterised the small-model regime and shown the direction of travel, not that we have located a threshold. Whether frontier-scale models invert at all is untested here.

Environments. ToolShed is our own construction. We built it because the central measurement needs counterfactual observations that no public benchmark provides, and

we have tried to be explicit about the ways that choice could flatter us: the perturbation operators are ours, the error messages are ours, and the task templates are ours. CodeRepair is the check on that, with MBPP problems, real mutations, and observations written by the Python interpreter rather than by us, and it shows the same pattern. Neither is a long-horizon, realistic agent benchmark. We would expect the effect to be larger, not smaller, in settings with longer transcripts and more failures per trajectory, but we have not shown that.

Perturbations are synthetic. Failing actions come from fixed operators rather than from model samples. This buys the property the scaling claim depends on, that every model is probed on byte-identical items, at the cost of some realism. Sampling failures from each model would make the item set model-dependent and would confound size with item difficulty; we judged that the worse trade. The operators were chosen to match the mistakes small models actually make, and the rollout study, where models generate their own failures, is where that choice is put to the test.

Greedy decoding. Rollouts use greedy decoding, so each (model, harness) pair yields one trajectory per task and our intervals over rollouts come from resampling tasks rather than seeds. Sampling would add a variance component we have not measured. Greedy is the right default for a comparison of harnesses, since it removes decoding noise from a contrast we care about, but it is not how every deployed agent runs.

Log-probabilities of whole strings. Corrective gain is a difference in summed token log-probability. Summed quantities grow with string length, so the raw nat values are not comparable across environments with very different action lengths. Within an environment they are comparable across conditions and models because the string being scored is identical; the normalised repeat probability and the exact

greedy repeat rate are provided precisely because they carry no such caveat.

A numerical failure worth knowing about. On this machine, calling `torch.set_num_threads(n)` with $n > 1$ causes the installed PyTorch build to return all-NaN logits, silently, with unchanged wall-clock time. We lost one complete probe run to this before noticing, and our first verification script failed to catch it because `max(0.0, nan)` returns `0.0` in Python, so a comparison of two NaN quantities reported perfect agreement. Model loading now ends with a forward pass that must produce finite logits, the scorer checks every returned logit row, and the verification script rejects non-finite and positive log-probabilities before comparing them. We record this because it is the kind of fault that produces a clean-looking table of fabricated numbers, and because the mitigation, which is to assert on the numerical path and not just on the code path, is cheap and general.

12 Broader impact

The interventions we study make small agents repeat themselves less, which reduces wasted tool calls, wasted tokens and the wall-clock cost of a stuck loop. The direct effects are mundane and mostly good. Two second-order points are worth naming. First, a harness that suppresses repeated failed actions makes an agent more persistent, and persistence is not always desirable: an agent that keeps trying new calls after several failures may do more damage in a side-effecting environment than one that gets stuck. Step budgets and side-effect confirmation remain necessary; nothing here replaces them. Second, our results make small models more usable as agents, which lowers the cost of running autonomous systems locally. That is broadly positive for privacy and access, and it also lowers the cost of running such systems without oversight.

13 Conclusion

Agent harnesses append a failed call and its error message to the transcript because the error is information. It is. But the transcript entry that carries it also carries the failed call’s exact token sequence, and for small models that second, unintended payload is much the larger of the two. Measured as a change in the log-probability of writing the failed call again, the net effect of the failure record is to make repetition *more* likely, on every model we tested, in both environments.

The decomposition is what makes this actionable rather than merely discouraging. The harm is carried by the surface form, not by a failure to read the message, so it is a property of the harness and not of the model, and harnesses are much cheaper to change than models. Keeping the diagnosis while withholding the string, or making the string unreachable at

decoding time, acts on the term that matters. Instructing the model not to repeat itself does not.

Neither does the remedy that sounds most obviously right. Deleting the failed attempt and retrying from a clean context is the worst harness we measured, because it restores precisely the context that produced the failure. The principle that survives all three comparisons is narrow enough to be useful: the context after a failure must differ from the context before it, and the difference must not be the failed action.

Three questions follow directly. Does the inversion persist at frontier scale, and if it disappears, does it do so because copying weakens or because the error message finally wins? Our decomposition answers that question with a few hundred forward passes per model, so it is cheap for anyone with the hardware. Second, how should a harness decide *what* to abstract? We used a fixed mapping from error family to gloss; a runtime that knows more about its own failures could say more, and there is a real trade-off between removing the hazard and removing the evidence. Third, can the effect be trained away rather than engineered around, in the way unlikelihood training and repetition penalties address open-ended text? Our results suggest what such a training signal would have to target, and it is not the model’s understanding of error messages.

References

- Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, et al. Smollm2: When smol goes big – data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025. URL <https://arxiv.org/abs/2502.02737>.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL <https://arxiv.org/abs/2108.07732>.
- Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. Small language models are the future of agentic ai. *arXiv preprint arXiv:2506.02153*, 2025. URL <https://arxiv.org/abs/2506.02153>.
- Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024. URL <https://arxiv.org/abs/2407.21787>.
- Louis Castricato, Nathan Lile, Suraj Anand, Hailey Schoelkopf, Siddharth Verma, and Stella Biderman. Suppressing pink elephants with direct principle feedback.

- arXiv preprint arXiv:2402.07896*, 2024. URL <https://arxiv.org/abs/2402.07896>.
- Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023. URL <https://arxiv.org/abs/2304.05128>.
- Nhi Hoai Doan, Tatsuya Hiraoka, and Kentaro Inui. Understanding and controlling repetition neurons and induction heads in in-context learning. *arXiv preprint arXiv:2507.07810*, 2025. URL <https://arxiv.org/abs/2507.07810>.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL <https://arxiv.org/abs/2407.21783>.
- Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019. URL <https://arxiv.org/abs/1904.09751>.
- Xinyi Hou, Shenao Wang, Yanjie Zhao, and Haoyu Wang. When agents do not stop: Uncovering infinite agentic loops in llm agents. *arXiv preprint arXiv:2607.01641*, 2026. URL <https://arxiv.org/abs/2607.01641>.
- Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023. URL <https://arxiv.org/abs/2310.01798>.
- Kyomin Hwang, Suyoung Kim, JunHoo Lee, and Nojun Kwak. Do not think about pink elephant! *arXiv preprint arXiv:2404.15154*, 2024. URL <https://arxiv.org/abs/2404.15154>.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SwEBench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. URL <https://arxiv.org/abs/2310.06770>.
- Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019. URL <https://arxiv.org/abs/1909.05858>.
- Jonggeun Lee, Woojung Song, Jongwook Han, Haesung Pyun, and Yohan Jo. Don’t adapt small language models for tools; adapt tool schemas to the models. *arXiv preprint arXiv:2510.07248*, 2025. URL <https://arxiv.org/abs/2510.07248>.
- Fangzheng Li, Aimin Zhang, and Chen Lv. Constraint tax in open-weight llms: An empirical study of tool calling suppression under structured output constraints. *arXiv preprint arXiv:2606.25605*, 2026. URL <https://arxiv.org/abs/2606.25605>.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023a. URL <https://arxiv.org/abs/2307.03172>.
- Xiao Liu, Hao Yu, Hanchen Zhang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023b. URL <https://arxiv.org/abs/2308.03688>.
- Aman Madaan, Niket Tandon, Prakhar Gupta, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. URL <https://arxiv.org/abs/2303.17651>.
- Callum McDougall, Arthur Conmy, Cody Rushing, Thomas McGrath, and Neel Nanda. Copy suppression: Comprehensively understanding an attention head. *arXiv preprint arXiv:2310.04625*, 2023. URL <https://arxiv.org/abs/2310.04625>.
- Grégoire Mialon, Clémentine Fourier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023. URL <https://arxiv.org/abs/2311.12983>.
- Catherine Olsson, Nelson Elhage, Neel Nanda, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022. URL <https://arxiv.org/abs/2209.11895>.
- Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023. URL <https://arxiv.org/abs/2305.15334>.
- Qwen, :, An Yang, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL <https://arxiv.org/abs/2412.15115>.
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. URL <https://arxiv.org/abs/2303.11366>.
- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024. URL <https://arxiv.org/abs/2408.03314>.

- Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. The limits of inference scaling through resampling. *arXiv preprint arXiv:2411.17501*, 2024. URL <https://arxiv.org/abs/2411.17501>.
- Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung yi Lee, and Yun-Nung Chen. Let me speak freely? a study on the impact of format restrictions on performance of large language models. *arXiv preprint arXiv:2408.02442*, 2024. URL <https://arxiv.org/abs/2408.02442>.
- Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjana Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*, 2024. URL <https://arxiv.org/abs/2407.18901>.
- Karthik Valmeekam, Matthew Marquez, and Subbarao Kambhampati. Can large language models really improve by self-critiquing their own plans? *arXiv preprint arXiv:2310.08118*, 2023. URL <https://arxiv.org/abs/2310.08118>.
- Tereza Vrabcová, Marek Kadlčík, Petr Sojka, Michal Štefánek, and Michal Spiegel. Negation: A pink elephant in the large language models’ room? *arXiv preprint arXiv:2503.22395*, 2025. URL <https://arxiv.org/abs/2503.22395>.
- Andrew Wang, Sophia Hager, Adi Asija, Daniel Khashabi, and Nicholas Andrews. Hell or high water: Evaluating agentic recovery from external failures. *arXiv preprint arXiv:2508.11027*, 2025. URL <https://arxiv.org/abs/2508.11027>.
- Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*, 2023. URL <https://arxiv.org/abs/2309.10691>.
- Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. *arXiv preprint arXiv:1908.04319*, 2019. URL <https://arxiv.org/abs/1908.04319>.
- Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023. URL <https://arxiv.org/abs/2307.09702>.
- Jin Xu, Xiaojiang Liu, Jianhao Yan, Deng Cai, Huayang Li, and Jian Li. Learning to break the loop: Analyzing and mitigating repetitions for neural text generation. *arXiv preprint arXiv:2206.02369*, 2022. URL <https://arxiv.org/abs/2206.02369>.
- An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL <https://arxiv.org/abs/2505.09388>.
- Zhanfu Yang. Why retrying fails: Context contamination in llm agent pipelines. *arXiv preprint arXiv:2605.08563*, 2026. URL <https://arxiv.org/abs/2605.08563>.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. URL <https://arxiv.org/abs/2210.03629>.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL <https://arxiv.org/abs/2406.12045>.
- Lu Yi, Runlin Lei, Liuyi Yao, Yuexiang Xie, Yuyang Li, Wenhao Zhang, Zhewei Wei, Yaliang Li, and Jian-Yun Nie. Learning agent-compatible context management for long-horizon tasks. *arXiv preprint arXiv:2605.30785*, 2026. URL <https://arxiv.org/abs/2605.30785>.
- Oussama Zenkri and Oliver Brock. Probing embodied llms: When higher observation fidelity hurts problem solving. *arXiv preprint arXiv:2605.20072*, 2026. URL <https://arxiv.org/abs/2605.20072>.
- 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. *arXiv preprint arXiv:2307.13854*, 2023. URL <https://arxiv.org/abs/2307.13854>.

A Reproduction

Runs excluded for being incomplete. Every model/environment cell reported here reached at least 30 scored items; none were excluded on that ground.

Environment. Python 3.12, PyTorch 2.4.0+cpu (CPU build), on Intel Core i7-6820HQ CPU @ 2.70GHz. No GPU is used or required. Exact pinned versions are in `requirements.txt` and `environment.yml`, and the software environment is recorded in `results/raw/**/*meta.json` for every run made after we started writing it there.

The Transformers version is not single-valued: the environment was rebuilt partway through the study, and runs exist under 4.45.0 and 4.56.1. Rather than assert that this does not matter, we measured it. `scripts/check_version_drift.py` re-scores stored rows in the current environment and compares them to what is on disk; over 210 (item, condition, candidate) triples for `smollm2-135m` the largest disagreement is 4.9×10^{-5} nats, roughly four orders of magnitude below the smallest effect the paper reports and consistent with float32 non-determinism in the kernel selection. Scoring is a deterministic forward pass, so this is the expected result; we record it because the alternative is asking the reader to take it on trust.

Pipeline.

```
python scripts/download_models.py --ladder
python scripts/build_items.py
python scripts/verify_scoring.py --model smollm2-135m
python scripts/launch_probes.py --phases ABC
python scripts/run_agent_study.py --models qwen2.5-0.5b \
    --env toolshed --n-tasks 30 --max-steps 8
python scripts/analyze_probe.py
python scripts/analyze_agent.py
python scripts/make_facts.py && python scripts/make_figures.py
```

Every stage is resumable. Probe results are keyed by (item, condition, candidate) and existing keys are skipped, so an interrupted run continues where it stopped; rollout files are written per (model, environment, harness, temperature, seed).

Seeds. All randomness derives from base seed 20260808 through named sub-streams (`slmecho.seeding.derive_seed`), so re-running one part of the study reproduces the same items even if the surrounding loop changed. World generation, task instantiation, item subsampling and the bootstrap each draw from their own stream.

Generation parameters. Probe scoring is teacher-forced and involves no sampling. Rollouts use greedy decoding with a 32-token cap per action, stopping at the chat template’s end-of-turn token or at a newline (tool calls only; program repair allows multi-line output with a 200-token cap).

B Probe conditions in full

Each row is one way of writing the same decision point into a transcript. The scored action string is identical across all of them; only the observation the harness records differs.

Condition	What the transcript contains
pre	context before the attempt (baseline)
fail	attempt + error observation (standard ReAct harness)
succ	attempt + counterfactual success observation
neut	attempt + valence-free observation
neut_pad	attempt + valence-free observation, length-matched to 'fail'
abstract	harness-generated failure description, no verbatim call
fail_echo	as 'fail', but the error quotes the failed call verbatim
fail_terse	as 'fail', with the shortest error rendering
fail_verbose	as 'fail', with the longest error rendering
fail_instr	as 'fail', plus an explicit do-not-repeat instruction
fail_k2	the same call fails twice
fail_k3	the same call fails three times
fail_early	as 'fail', but the failure precedes the successful steps
fail_other	a DIFFERENT wrong action failed; the scored one was never written
abstract_min	bare failure marker, no diagnosis and no verbatim call

Table 10: Every probe condition. All conditions for one item share the same system prompt, goal and successful prefix, byte for byte.

C Perturbation operators

Failing calls are not sampled from a model, they are constructed, so that the failure is of a known kind and the correct call is known exactly. Each operator takes a valid call and breaks it in one specific way. Examples are shown as they appear in the transcript, truncated where they run past the column.

Operator	Reference call	Perturbed call	Error family
arg_typo	find_contact (name='Dana Bergmann')	find_contact (contact_name='Dana Bergmann')	unexpected_argument
date_format	list_events (date='2026-09-19')	list_events (date='September 19, 2026')	format_error
drop_arg	send_message (to='dana.bergmann@corp.example', subject='agenda', body='roadmap', hiring, i...	send_message (to='dana.bergmann@corp.example', subject='agenda')	missing_argument
hallucinated_entity	find_contact (name='Dana Bergmann')	find_contact (name='Alex Ramirez')	contact_not_found
name_for_email	send_message (to='dana.bergmann@corp.example', subject='agenda', body='roadmap', hiring, i...	send_message (to='Dana Bergmann', subject='agenda', body='roadmap', hiring, incident review...)	format_error
readonly_target	write_file (path='reports/roster.md', content='Elena Okafor, Karim Okafor')	write_file (path='system/config.ini', content='Elena Okafor, Karim Okafor')	permission_error
tool_typo	find_contact (name='Dana Bergmann')	find_contacts (name='Dana Bergmann')	unknown_tool
type_swap	create_event (title='1:1', date='2026-10-13', time='09:30', attendees=['karim.silva@corp...])	create_event (title='1:1', date='2026-10-13', time='09:30', attendees=['karim.silva@corp...])	type_error

Table 11: Perturbation operators used to build failing actions in ToolShed, with one instantiated example each.

D Per-error-family results

The same headline quantity, split by the kind of error the failing call provoked. The point of the table is the absence of structure: if the effect were an artefact of one error message’s wording, one row would carry it.

Model	Params	p_{rep} before	p_{rep} after	Δp_{rep}	greedy before	greedy after	greedy abs.
SmolLM2-135M	0.14B	0.111 [0.053, 0.180]	0.951 [0.917, 0.978]	0.840 [0.770, 0.901]	0.000 [0.000, 0.000]	0.000 [0.000, 0.000]	0.000 [0.000, 0.000]
SmolLM2-360M	0.36B	0.089 [0.032, 0.156]	0.856 [0.795, 0.910]	0.767 [0.689, 0.840]	0.000 [0.000, 0.000]	0.100 [0.033, 0.183]	0.000 [0.000, 0.000]
Qwen2.5-0.5B	0.49B	0.075 [0.023, 0.138]	0.833 [0.766, 0.893]	0.758 [0.680, 0.831]	0.000 [0.000, 0.000]	0.017 [0.000, 0.050]	0.000 [0.000, 0.000]

Table 13: Repetition probabilities on coderepair. p_{rep} is the normalised probability of the failed call among the scored candidate actions; *greedy* is the fraction of items on which greedy decoding reproduces the failed call exactly.

Error family	SmolLM2-135M	SmolLM2-360M	Qwen2.5-0.5B	Qwen3-0.6B	Llama-3.2-1B	SmolLM2-1.7B
permission_error	-44.4 [-47.0, -42.2]	-43.0 [-44.9, -41.3]	-37.9 [-42.4, -33.9]	-51.9 [-52.9, -51.1]	—	-30.1 [-33.1, -27.4]
contact_not_found	-34.4 [-36.4, -31.3]	-21.9 [-23.0, -20.5]	-27.8 [-30.9, -26.1]	—	—	-19.0 [-21.3, -17.5]
format_error	-22.9 [-24.5, -21.1]	-18.4 [-20.1, -16.9]	-15.1 [-17.8, -12.4]	-23.9 [-27.5, -19.9]	-14.7 [-18.2, -11.6]	-13.2 [-14.5, -12.1]
file_not_found	—	—	—	—	—	-16.9 [-18.7, -14.1]
event_not_found	-18.7 [-19.2, -18.4]	-13.0 [-13.7, -12.3]	-11.7 [-12.4, -11.2]	—	—	—
missing_argument	-17.3 [-20.2, -13.6]	-15.4 [-18.6, -11.5]	-10.4 [-12.0, -8.3]	-22.7 [-24.7, -19.3]	-9.0 [-15.0, -1.1]	-9.8 [-12.1, -7.2]
unknown_tool	-18.6 [-22.4, -15.4]	-13.5 [-18.9, -9.0]	-9.8 [-12.3, -7.4]	-16.1 [-19.9, -13.2]	-11.0 [-15.4, -7.5]	-12.2 [-14.6, -9.9]
attendee_not_found	—	—	—	—	—	-12.3 [-12.9, -11.1]
unexpected_argument	-19.0 [-22.9, -14.8]	-12.3 [-16.4, -8.8]	-5.9 [-10.2, -1.1]	-9.1 [-13.4, -5.5]	-8.6 [-11.7, -5.8]	-10.1 [-11.4, -8.8]
type_error	-18.9 [-19.6, -18.2]	-11.2 [-12.6, -9.8]	-0.7 [-3.2, 1.8]	-6.3 [-11.8, -1.3]	-3.2 [-4.0, -2.3]	-7.7 [-10.2, -5.3]

Table 12: Corrective gain (nats) by error family on ToolShed, sorted by pooled mean. Rows are the error the failing call provoked; a broadly uniform column argues against any explanation resting on the wording of one error type.

E Repetition statistics on program repair

The main text reports the repetition readouts for the simulated tool environment; the same quantities for MBPP program repair are below. They are computed identically, over the same fixed candidate set construction.

F Prompts

The complete system prompts, the failure-abstraction template, and the “do not repeat” instruction are reproduced verbatim below; they are also emitted by `scripts/dump_prompts.py` so they cannot drift from the code. Two typesetting conventions apply and nothing else has been altered: a line too long for the page is broken and the broken line ends in a backslash, and the few non-ASCII characters the prompts contain (an em dash, curly quotes) are transliterated, since the typewriter font has no glyph for them.

ToolShed system prompt (probe)

You are a tool-using assistant working inside a small office workspace. Solve the user’s request by \ calling tools one at a time.

Rules:

- Reply with exactly one tool call and nothing else.
- Use the form `tool_name(arg=value)`, with keyword arguments.
- Dates are YYYY-MM-DD and times are HH:MM.
- When the request is fully satisfied, reply with `finish()`.

Available tools:

- `list_files(folder: str) -> list[str]`
List the files inside a folder.
folder: folder name, e.g. ‘notes’
- `read_file(path: str) -> str`
Return the contents of a file.
path: full path, e.g. ‘notes/todo.md’
- `write_file(path: str, content: str) -> str`
Create or overwrite a file.
path: full path
content: text to store
- `delete_file(path: str) -> str`
Delete a file.
path: full path
- `find_contact(name: str) -> dict`
Look up one contact by full name.
name: exact full name
- `list_contacts(team: str) -> list[dict]`
List the members of a team.

```

    team: team name, e.g. 'design'
- list_events(date: str) -> list[dict]
    List calendar events on a date.
    date: date as YYYY-MM-DD
- create_event(title: str, date: str, time: str, attendees: list[str]) -> str
    Add a calendar event.
    title: event title
    date: date as YYYY-MM-DD
    time: start time as HH:MM
    attendees: list of attendee emails
- cancel_event(event_id: str) -> str
    Cancel a calendar event by id.
    event_id: event id, e.g. 'evt-4'
- send_message(to: str, subject: str, body: str) -> str
    Send a message to one recipient email address.
    to: recipient email address
    subject: subject line
    body: message body
- set_reminder(text: str, date: str) -> str
    Store a reminder for a date.
    text: reminder text
    date: date as YYYY-MM-DD
- convert_units(value: float, from_unit: str, to_unit: str) -> float
    Convert a value between two units.
    value: numeric value
    from_unit: source unit
    to_unit: target unit

```

Format demonstration appended in the agent study

A valid reply looks exactly like this, with no prefix or explanation:
`list_files(folder='notes')`

CodeRepair system prompt

You are a Python programmer. The user gives a task and the tests that will be run against your code.

Rules:

- Reply with one Python function definition and nothing else.
- Do not include the tests, comments, explanations or markdown fences.
- Use exactly the function name that appears in the tests.

Do-not-repeat instruction

Important: some of your earlier tool calls failed. Do not repeat a call that has already failed; change \ the call before trying again.

Failure abstraction (example)

[attempt 1 failed: find_contact --- an argument name was not recognised; that call is not repeatable]

Error-family glosses used by the abstraction

arity_error	-> wrong number of arguments
attendee_not_found	-> no such attendee
contact_not_found	-> no such contact
event_not_found	-> no such event
file_not_found	-> no such file
folder_not_found	-> no such folder
format_error	-> an argument was badly formatted
missing_argument	-> a required argument was missing
parse_error	-> the call could not be parsed
permission_error	-> the target was read-only
recipient_not_found	-> no such recipient
runtime_error	-> the code raised an exception
state_error	-> the target was already in the requested state
syntax_error	-> the code did not compile
team_not_found	-> no such team
test_failure	-> the code did not pass the tests
type_error	-> an argument had the wrong type
unexpected_argument	-> an argument name was not recognised
unknown_tool	-> no such tool
unsupported_conversion	-> that conversion is not supported

G Example trajectories

One task under every harness, chosen by rule: the task with the most exact repeats under the standard harness. It is a single task and it is the worst one, so it illustrates rather than demonstrates, and the quantitative claims are in Section 8, but it is worth reading for what it shows and also for what it complicates.

Under *verbatim*, *drop* and *abstract* the model writes the same call six times. Under the two ban harnesses it cannot, and what it produces instead is the same call with material accreted onto it: fragments of the error message under *verbatim+ban*, chained `.replace` calls under *abstract+ban*. Neither is a reconsidered call. *verbatim+instr* is the one that behaves differently here, trying four distinct calls, none of them right.

Two of those cut against the aggregates and we would rather show that than pick a friendlier task. The abstraction reduces repetition across the task set but does nothing on this item, and the instruction does nothing across the task set but changes the trajectory on this one. Task-level variance at this scale is large, which is why the paper’s claims rest on paired comparisons over all tasks rather than on examples.

qwen2.5-0.5b, harness=verbatim

```
goal: agent-0011 solved=False
1. [ok ] convert_units(value=88, from_unit='km', to_unit='mi')
2. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
3. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
4. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
5. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
6. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
```

qwen2.5-0.5b, harness=verbatim-instr

```
goal: agent-0011 solved=False
1. [ok ] convert_units(value=88, from_unit='km', to_unit='mi')
2. [ERR] read_file(path='observations/54.6806.txt')
3. [ERR] delete_file(path='observations/54.6806.txt')
4. [ERR] list_files(folder='observations')
5. [ok ] list_files(folder='drafts')
6. [ok ] list_files(folder='drafts')
```

qwen2.5-0.5b, harness=drop

```
goal: agent-0011 solved=False
1. [ok ] convert_units(value=88, from_unit='km', to_unit='mi')
2. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
3. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
4. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
5. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
6. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
```

qwen2.5-0.5b, harness=abstract

```
goal: agent-0011 solved=False
1. [ok ] convert_units(value=88, from_unit='km', to_unit='mi')
2. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
3. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
4. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
5. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
6. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
```

qwen2.5-0.5b, harness=verbatim-ban

```
goal: agent-0011 solved=False
1. [ok ] convert_units(value=88, from_unit='km', to_unit='mi')
2. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
3. [ERR] write_file(path='reports/conversion.md', content=str(54.6806), error='write_file: argument value ~
4. [ERR] write_file(path='reports/conversion.md', content=str(54.6806), error='write_file: SyntaxError: co~
5. [ERR] write_file(path='reports/conversion.md', content=str(54.6806), error='write_file: SyntaxError: co~
6. [ERR] write_file(path='reports/conversion.md', content=str(54.6806), error='write_file: argument value ~
```

qwen2.5-0.5b, harness=abstract-ban

```
goal: agent-0011 solved=False
1. [ok ] convert_units(value=88, from_unit='km', to_unit='mi')
2. [ERR] write_file(path='reports/conversion.md', content=str(54.6806))
3. [ERR] write_file(path='reports/conversion.md', content=str(54.6806).replace('.', ','))
4. [ERR] write_file(path='reports/conversion.md', content=str(54.6806).replace('.', ',').replace('-', ','))
5. [ERR] write_file(path='reports/conversion.md', content=str(54.6806).replace('.', ',').replace('-', ','))
6. [ERR] write_file(path='reports/conversion.md', content=str(54.6806).replace('.', ',').replace('-', ','))
```