How Many Parameters Is a Trace Worth? Measuring Borrowed Capability in LLM Agents

Anonymous authors
Paper under double-blind review

Abstract

Language-model agents rarely operate in a vacuum, as they typically have access to previous requests and execution traces. These traces are especially valuable for systems that rely on small local models, where traces produced by a larger model can guide a smaller, faster one. In this paper, we study *borrowed capability*: the degree to which a weaker model, given a stronger model's verified execution trace on a related task, can perform as if it had the stronger model's capability. We derive 1,200 targets from 300 source tasks across six agentic benchmarks; each target changes the source's arguments, context, procedure, or goal, so that replaying the source trace fails. Across 20 open checkpoints from four families (270M–34B), each attempting every target with and without the trace, we find that a trace is worth roughly doubling the executor's size, provided that the executor is competent in the environment, i.e., solves at least one of its original tasks on its own. For competent executors, trace access raises exact success by 12 points, about as much as replacing the executor with a same-family model twice its size. Within an environment, larger executors close a larger share of the gap to the trace's author, although the smallest ones gain the most points. Executors below this competence threshold gain only 1.5 points, unless what they fail is the required answer format, which the trace demonstrates. Traces help least when the interface has changed, and there they can mislead. A trace, in short, can make a small agent act like a larger one, but it cannot teach an agent to act.

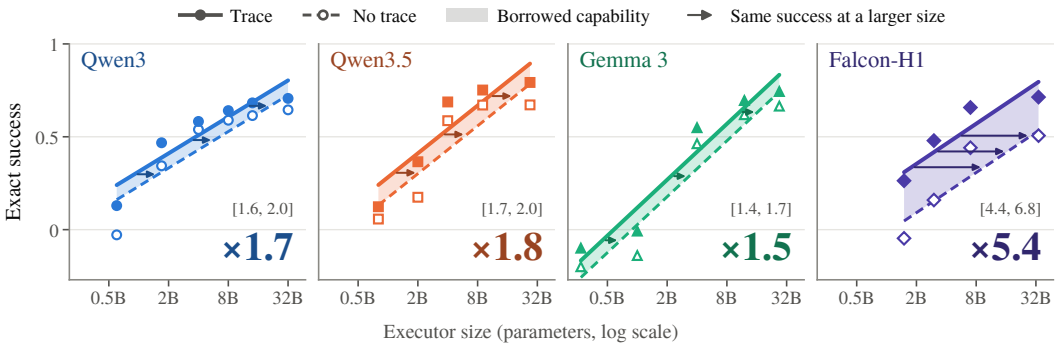

Figure 1: A trace is worth roughly doubling the executor. Exact success of competent executors against size, per family, with (solid) and without (dashed) the trace, adjusted for environment; points are checkpoint means. Under a fit with parallel curves, the trace shifts each curve by a constant factor: a traced executor performs like an unaided one larger by the factor shown (arrows; 95% intervals in brackets; 2.0 pooled).

1 Introduction

The capability of a language model is, to a first approximation, dependent on its number of parameters (Kaplan et al., 2020). Agentic deployments, however, produce a second resource as a byproduct of their ordinary operation: execution traces. Production agents often perform specialized tasks repetitively and with little variation (Belcak et al., 2025), so a deployed system operates on a growing archive of verified solutions to tasks close to the ones it will see next. While the

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

value of parameters has been characterized in considerable detail (Hoffmann et al., 2022; Wei et al., 2022a; Ruan et al., 2024), the value of a trace has not.

The question is most consequential in local, on-device deployments. Deployments that pair a small local model with a stronger remote one have become consumer defaults (Gunter et al., 2024; Gemini Team, 2023; Microsoft, 2024), serving stacks route requests between models of different sizes to control cost (Chen et al., 2023; Ong et al., 2024), and local-remote protocols delegate execution to on-device models under cloud supervision (Narayan et al., 2025). In each of these architectures, the small model performs most of the work, while the larger model's solutions remain available to the system.

For example, consider an on-device assistant asked to file this month's expense report, a few weeks after a stronger remote model filed the previous one by collecting the receipts from email, entering them in the finance spreadsheet, and sending the total to the user's manager. The old trace cannot simply be replayed, since the receipts, amounts, and dates have all changed. Nevertheless, it exposes information that generic prompting does not: where receipts arrive, which sheet and columns the finance team uses, who must be notified, and one complete route from request to success. If the trace can guide the smaller model, the cost of invoking the larger one is paid once and amortized across every subsequent report. If, instead, the smaller model follows the trace too literally, the same context can mislead it. We refer to the quantity at stake as **borrowed capability**: the degree to which a weaker model, given a stronger model's verified execution trace on a related task, can perform as if it had the stronger model's competence.

Whether capability can be borrowed, however, has so far been assumed rather than measured. Systems that store and reuse experience (Shinn et al., 2023; Zheng et al., 2024; Wang et al., 2025b; Zhao et al., 2024a; Wei et al., 2025; Zhang et al., 2025b) entangle at least four effects: what experience is stored, which memory is retrieved, how it is represented, and whether the executor can adapt it. Notably, Narayan et al. (2025) identify the failure of small local models to follow a remote model's multi-step guidance as a key bottleneck of local-remote systems. The field thus lacks a controlled measurement of the value of a single verified trace as a function of the two variables that govern it: the executor that must use the trace and how the new task differs from the traced one. Our question is thus:

Research Question

To what degree can a weaker model, given a stronger model's verified trace of a related task, complete the new task?

In this paper, we measure borrowed capability through a paired comparison on six agentic benchmarks (Trivedi et al., 2024; Liu et al., 2023; Yang et al., 2023; Patil et al., 2025; Li et al., 2023; Deng et al., 2023).[1] Starting from 300 source tasks, each solved once by a frontier author, we construct four targets per source, which change, respectively, its arguments, its context, its procedure, or its goal. We design the tasks so that replaying the source trace fails every target, so any gain must come from adaptation rather than copying. Each executor attempts every target twice, with and without the source trace, all else held fixed. We express the value of a trace through three measures: the resulting gain in success, the share of the gap to the author that it closes, and in parameters, i.e., as the size of an unaided model of the same family that would perform as well. We evaluate 20 open checkpoints from four families (namely Qwen3, Qwen3.5, Gemma 3, and Falcon-H1), spanning 270M–34B parameters.

Our results show that a trace is worth roughly doubling the executor's size (Figure 1), provided that the executor is **competent** in the environment, i.e., it solves at least one of the environment's original tasks on its own. Below this **competence threshold**, a trace is worth almost nothing, even though it demonstrates a closely related task. The one exception is executors whose only visible failure is the answer format: they never answer in the exact JSON format that the benchmark requires, and the trace shows them how. Among competent executors, larger ones close a larger share of the gap to the author within an environment, although the smallest ones gain the most; pooling environments hides this trend, since larger executors are also competent in harder environments. Finally, traces

[1] Our source code is available anonymously at `https://anonymous.4open.science/r/trace-adaptability`.

help least when the interface has changed, where they can even mislead the executor. In short, traces and parameters are substitutes above the competence threshold, but not below it.

Our contributions are the following:

- **A controlled measurement of borrowed capability.** We design a paired protocol that isolates the effect of a single verified trace from storage, retrieval, and representation, and expresses its value through three measures: the gain, the recovery rate, and the parameter-equivalence;
- **An exchange rate between traces and parameters.** Across 20 checkpoints and six environments, we show that a trace is worth roughly doubling a competent executor and almost nothing below the competence threshold, and we characterize how its value depends on scale and on the type of change;
- **Open tasks and traces.** We release the 300 source tasks, their verified traces, and 1,200 reviewed targets, with the runtime required to reproduce every paired measurement.

Overall, as more of deployed agents' work shifts to small local models, we argue that how much capability they can borrow, not just how much they inherently have, will increasingly determine their effectiveness.

2 Measuring the Value of a Trace

Tasks, traces, and executors. An agentic *task* consists of a natural-language request, an initial environment state, and a deterministic checker that inspects the terminal state and returns an exact success $Y \in \{0, 1\}$ and a partial score $Q \in [0, 1]$. An *execution trace* is the observable sequence of actions and tool or environment outputs produced while attempting a task, excluding hidden reasoning; it is *verified* if it passes the checker. We call the model that produced a trace its *author*, and the model that attempts a new task with the trace in context the *executor*. Each source task i has a verified trace τ_i and four targets (i, d), where $d \in \{\text{D1}, \ldots, \text{D4}\}$ is the type of change (Section 3). For a checkpoint m, $Y^z_{m,i,d}$ denotes exact success with ($z = 1$) or without ($z = 0$) access to τ_i; each model, target, and condition receives one attempt.

Definition 2.1 (Gain)**.** For a set of target cells $\mathcal{I}$, the *gain* of checkpoint m is the paired difference in exact success,

$$\widehat{\Delta}_m(\mathcal{I}) = \frac{1}{|\mathcal{I}|} \sum_{(i,d)\in\mathcal{I}} \left(Y^1_{m,i,d} - Y^0_{m,i,d}\right). \tag{1}$$

Since the two conditions differ only in access to the trace, the gain is the causal effect of that access under our setup. We note that $\widehat{\Delta}_m(\mathcal{I})$ can be negative (i.e., a trace hurts performance). Replacing Y with Q gives the partial-score gain.

The gain alone does not show how large an improvement is relative to what is achievable.

Definition 2.2 (Recovery rate)**.** With $\bar{Y}^0_m(\mathcal{I})$ the executor's unaided success and $\bar{S}_A(\mathcal{I})$ the author's first-attempt success on the same targets, the *recovery rate*

$$R_m(\mathcal{I}) = \frac{\widehat{\Delta}_m(\mathcal{I})}{\bar{S}_A(\mathcal{I}) - \bar{Y}^0_m(\mathcal{I})} \tag{2}$$

is the share of the gap between author and executor that the trace closes.

We compute it as a ratio of averages over a group of targets, and do not report it when the gap is below 0.05, where it becomes unstable.

To state the value of a trace in deployment-relevant units, we ask how large an unaided member of the executor's family would have to be to match the traced executor.

Definition 2.3 (Parameter-equivalence)**.** Let $g^z_f(s)$ be the success of family f at parameter count s in condition z. The *parameter-equivalence* of a trace for an executor of size s is

$$P_f(s) = s^\star/s, \qquad \text{where } g^0_f(s^\star) = g^1_f(s). \tag{3}$$

For example, $P_f = 2$ means that the trace lets the executor perform like a family member twice its size. Since each family has only four to six checkpoints, we fit $g^z_f(s) = \alpha_f + \beta_f \log s + \gamma_f z$, for

which Eq. 3 gives $P_f = \exp(\gamma_f/\beta_f)$ at every size; pooled fits share β and γ across families. We fit it to each checkpoint's mean success in each environment where it is competent (see below), with environment intercepts. This is because larger checkpoints are competent in harder environments, and without these intercepts a traced executor would be compared with smaller unaided ones on an easier mix of environments. Following Le Scao & Rush (2021), we keep the conversion deliberately within-family. We stress that P_f is a performance equivalence, not a causal effect of scale or cost equivalence, since a trace occupies context that the larger model would not need.

Finally, a trace can only help an executor that is able to operate the environment in the first place. Every executor thus also attempts every source once, without any trace.

Definition 2.4 (Competence)**.** An executor is *competent* in an environment if it solves at least one of the environment's source tasks unaided.

Since the sources are distinct from the targets and are run separately, this criterion is fixed before, and independently of, any trace; it separates executors that cannot operate an environment at all from those that can, but lack the capability for a particular task. We refer to it as the *competence threshold*, and report a stricter threshold of five sources as a sensitivity analysis.

Research questions. We organize our study around three research questions.

RQ1 (Value of a trace). How much is a verified trace worth, in terms of success and in model scale? *Rationale:* If a trace is equivalent to multiplying scale, capability for repetitive workloads can be amortized: a task is solved once by a strong model and executed thereafter by small ones.

RQ2 (Does every model benefit?). Which executors can use a trace, and how does the share of the gap they close depend on their scale? *Rationale:* Adapting a trace is itself a task. If it requires capability of its own, the executors that need a trace the most may be the ones that can use it the least.

RQ3 (When it fails). Where does a trace stop helping, or mislead, and why? *Rationale:* A trace that no longer fits can be misleading; locating these cases determines when a deployed system should stop relying on existing traces.

3 Tasks, Conditions, and Executors

Source tasks and their verified traces. We draw 50 source tasks from each of six benchmarks covering complementary interfaces: multi-application API workflows (AppWorld; Trivedi et al., 2024), shell interaction (AgentBench OS; Liu et al., 2023), iterative SQL (InterCode; Yang et al., 2023), typed function calling (BFCL; Patil et al., 2025), API selection in dialogue (APIBank; Li et al., 2023), and grounded web actions (Mind2Web; Deng et al., 2023). In the first three, the model acts on an isolated environment whose final state is scored; in the last three, a stateless response is scored. Sources were selected deterministically from an audit of 4,221 candidates to approximate each benchmark's distribution, without consulting any evaluated model. During design, we identified mistakes and issues with checkers that prevented accurate scoring; we therefore report scores under our versioned checkers rather than upstream leaderboard scores (Appendices A and D). Each source has one verified trace from a single frontier author, GPT-5.6 Sol at `xhigh` reasoning. We allow up to five attempts to generate a trace that passes the checker; for 12 cases with no successful trace, we replaced the tasks with new ones in a predeclared order. Traces contain only observable events, with credentials replaced by placeholders, and are shared by all targets of a source and by all executors.

Adaptation tasks. Rather than defining a numeric distance between tasks, we construct one target of each of four types per source:

- **D1 (Parameter change):** same goal, context, and procedure, with different material arguments (e.g., entities, dates, predicates);
- **D2 (Context change):** same goal and procedure, but part of the operating context has changed (e.g., a tool or database schema, a page layout), so that the trace no longer maps one-to-one onto the environment;

- **D3 (Procedure change):** an analogous goal that requires inserting, removing, reordering, or conditioning a material step;
- **D4 (Goal change):** a nearby outcome for which knowledge or a subprocedure of the source remains useful.

For instance, for an AppWorld source that marks as read every email thread from before the last calendar month, the D1 target changes the cutoff, the D2 target exposes the same request through a reorganized SDK, the D3 target marks the qualifying inbox threads as read and archives the outbox ones, and the D4 target archives the qualifying unread threads instead. We stress that the types are categorical, not ordinal; further examples are reported in Appendix B. Each target was authored item by item by coding agents running GPT-5.6 Sol, i.e., the trace author, under a rubric and release criteria that we fixed, to be natural, of the claimed type, roughly as difficult as its source, and natively solvable, and so that replaying the source trace fails for a material reason (Appendix C). The author solved 1,197 of the 1,200 targets within five attempts, and its first attempts provide $\bar{S}_A$ (Appendix I).

Paired conditions: with and without the trace. In TRACE, the executor receives the source request and its verified trace, framed as a successful execution of a related task, followed by the target request; in NOTRACE, it receives the native target presentation only. Target, environment, tools, scorers, checkpoint, decoding, and budgets are otherwise identical. Five routes deliver the complete trace inline. Since AppWorld observations can be hundreds of kilobytes long, AppWorld instead delivers a shortened trace, computed from the source alone: author actions are kept verbatim, long observations are condensed, and the executor can read the omitted content through two optional tools (Appendix E). For APIBank's context-change targets, headline estimates use a presentation revised after the main collection, which shows the trace as labeled history followed by the complete current task; Section 4.3 explains why, and Table 8 reports estimates under the original presentation.

Executors, runtime, and statistics. We evaluate 20 instruction-tuned checkpoints: Qwen3 (0.6B, 1.7B, 4B, 8B, 14B, 32B), Qwen3.5 (0.8B, 2B, 4B, 9B, 27B), Gemma 3 (270M, 1B, 4B, 12B, 27B), and Falcon-H1 (1.5B, 3B, 7B, 34B) (Yang et al., 2025; Qwen Team, 2026; Gemma Team, 2025; Zuo et al., 2025), which we group into tiers below 2B, 2–9B, and above 9B. Revisions and serving artifacts are frozen, with one deviation that changes no conclusion (Appendix F). Each cell receives one attempt at temperature zero, with thinking enabled for Qwen checkpoints, serial tool calls, no automatic retries, and route-specific budgets (Appendix G). The design has 48,000 target cells and 6,000 unaided source runs. Intervals are 95% percentile intervals from 10,000 bootstrap draws, which resample the 50 sources of each environment together with their targets and both conditions and are shared across checkpoints; scale trends are least-squares slopes of recovery on $\log_2$ parameters within a family and environment (Appendix H).

4 RESULTS

Unless stated otherwise, estimates cover the 89 of 120 checkpoint–environment blocks in which the executor is competent (17,800 pairs of targets). Appendix K reports how each estimate changes under alternative analysis choices (Table 8), as well as results per checkpoint, environment, and family.

4.1 THE VALUE OF A TRACE (RQ1)

Overall gain. Competent executors benefit substantially from a trace: trace access raises their exact success from 48.6% to 60.5%, a gain of 11.9 points (95% CI 10.7–13.1) that closes about a quarter of the gap to the author (Table 1). The gain is positive for every checkpoint, family, and environment, as well as for every type of change, although it is largest when only the arguments change and smallest when the context does (Section 4.3).

Exchange rate. Converted into parameters (Figure 1), the same gain is worth roughly doubling the executor: the pooled multiplier is 2.0 (95% CI 1.9–2.2), and 1.5–1.8 for Qwen3, Qwen3.5, and Gemma 3. Falcon-H1 is an outlier (5.4), since its unaided curve is shallow and its gains are the largest. This exchange rate is stable across analysis choices, ranging from 1.6 to 2.6 (Table 8). In

Table 1: Exact success without and with the trace (%) and gain (points), by whether the executor is competent in the environment. Blocks are checkpoint–environment combinations; "format only" are six Mind2Web blocks in which every untraced answer fails the required output format. The last column is each regime's share of the net gain.

Regime	Blocks	Pairs	Unaided	Traced	Gain [95% CI]	Share
Competent	89	17,800	48.6	60.5	11.9 [10.7, 13.1]	71.7%
Not competent	25	5,000	0.1	1.6	1.5 [1.2, 1.9]	2.6%
Format only	6	1,200	0.0	63.3	63.3 [60.8, 65.7]	25.7%
All	120	24,000	36.1	48.4	12.3 [11.4, 13.2]	100.0%

other words, for a competent executor, a single verified trace can stand in for a model twice its size, at the cost of context rather than parameters.

Capability, not format. While one may argue that a trace mostly shows the executor the expected output format, this explanation only accounts for a small part of the gain. Excluding every pair in which either attempt produced a malformed answer retains 80% of the gain and a multiplier of 1.8. Moreover, the gain is on tasks that the executor could not solve: across all blocks, on parameter changes whose source the executor failed unaided, success rises from 12.5% to 35.7% (Appendix L).

Key Observation #1

For competent executors, a verified trace is worth roughly doubling the executor's size.

4.2 Who Benefits from a Trace (RQ2)

Across all 24,000 pairs, the gain of 12.3 points is due to three regimes whose gains differ by an order of magnitude (Table 1): competent executors, executors below the competence threshold, and six blocks whose only obstacle is the answer format.

Below the competence threshold. Executors that are not competent in an environment gain only 1.5 points (CI 1.2–1.9), even though every trace demonstrates a closely related task, and most of this gain comes from a single checkpoint that mostly fails Mind2Web's answer format (Appendix M). Notably, size alone does not guarantee competence: no Gemma 3 or Falcon-H1 checkpoint, up to 34B, solves an AppWorld source, and none gains more than one point there, whereas the seven competent checkpoints gain 17 points. This suggests that a trace cannot replace the ability to operate an environment: an executor must be able to act on its own before it can borrow from the actions of another model.

The format exception. Six Mind2Web blocks, namely the three largest Gemma 3 and Falcon-H1 checkpoints, solve nothing unaided, because every answer violates the required format: the benchmark expects a single bare JSON object, whereas, e.g., Falcon-H1 wraps it in a Markdown code block. With the trace, whose answer is bare JSON, they succeed on 63% of the targets. At least some of these executors were already capable: 68% of Falcon-H1 7B's enclosed answers are correct, against 87.5% with the trace (Appendix M). In other words, the larger Falcon-H1 checkpoints could already solve most of these tasks, and the trace mostly shows them the required answer format. Since they are only 5% of the pairs but a quarter of the net gain, we report them separately.

Scale. Among competent executors, scale plays two opposite roles (Figure 2). Within an environment, larger executors close a larger share of the gap to the author: recovery rises with size in 15 of the 16 family–environment combinations where the trend can be estimated (11 with intervals excluding zero), by a median of 7 points of the gap per doubling (Appendix N). Smaller executors, however, start further from the author, so they gain the most points while closing the smallest share of the gap (19.4 points below 2B, against 10.1 above 9B). In other words, the executors that use a trace the least efficiently are also those that benefit from it the most.

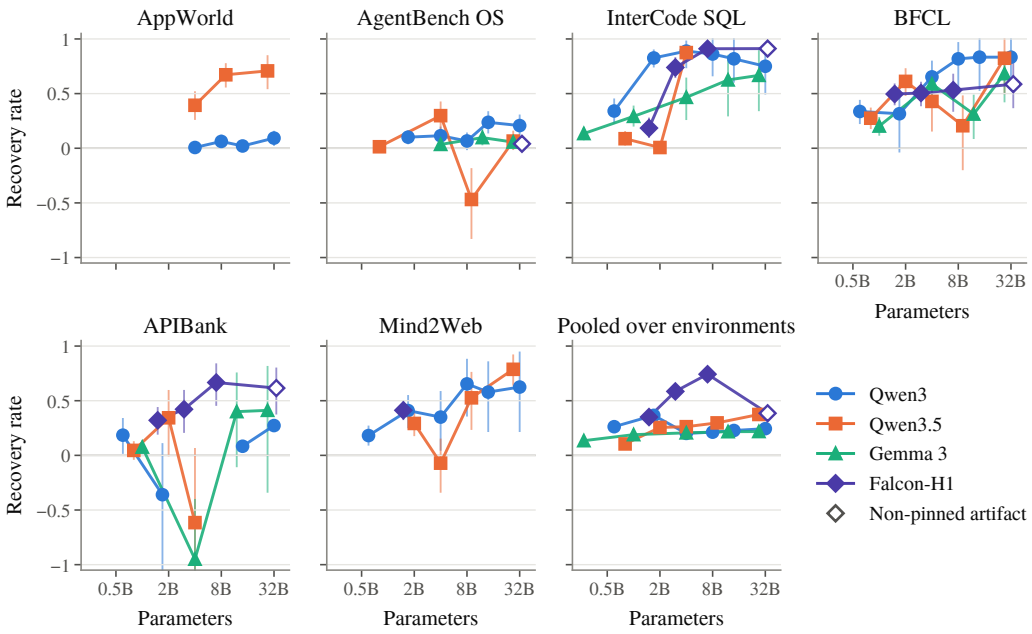

Figure 2: Recovery rate on parameter, procedure, and goal changes against executor size, per environment and family, for competent executors, with 95% intervals; sizes whose gap to the author is below 0.05 are omitted. The last panel pools each checkpoint's environments (point estimates).

Pooling hides the trend. Pooled across environments, recovery barely moves with size (24%, 31%, and 28% by tier), because larger executors are also competent in the long-horizon environments, where recovery is lowest. In Qwen3, for instance, recovery rises with size in each of the five environments where it can be estimated, yet drops when AppWorld enters the pool at 4B (Figure 2, last panel).

Key Observation #2

A trace helps only competent executors and, within an environment, the larger the executor, the larger the share it borrows.

4.3 When Traces Mislead (RQ3)

Context changes. Traces help least when the interface has changed. Competent executors gain 6.4 points on context changes, and only 1.6 points (CI −0.7 to 4.0) outside APIBank; moreover, the trace turns a success into a failure on 9.1% of these targets, against 3.3–5.2% for the other types (Figure 3, left). The harm can be systematic: on InterCode SQL, where the executor must discover the changed schema by querying the database, Qwen3 is harmed at every size from 4B to 32B, whereas Qwen3.5 checkpoints of similar size are not. In other words, a trace that no longer maps onto the environment is not merely less useful, as it can actively mislead the executor.

Presentation. APIBank offers the clearest case: its context-change targets rename exactly one parameter and state that field names from the source interface are unavailable. Under our original presentation, in which the trace and its obsolete interface follow the current one, success across all 20 checkpoints falls from 52% unaided to 6% with the trace, and, in an audit of nine checkpoints, 72% of traced failures differ from the correct call only by the obsolete field (Appendix P). Presenting the same trace as a labeled historical block, followed by the complete current task, raises traced success to 75% (58% with the history placed last; Figure 3, right). In other words, the same trace can mislead an executor or help it, depending on how it is presented. We stress that the labeled presentations also change the roles of the messages, so we do not attribute this difference to a single element (Appendix O).

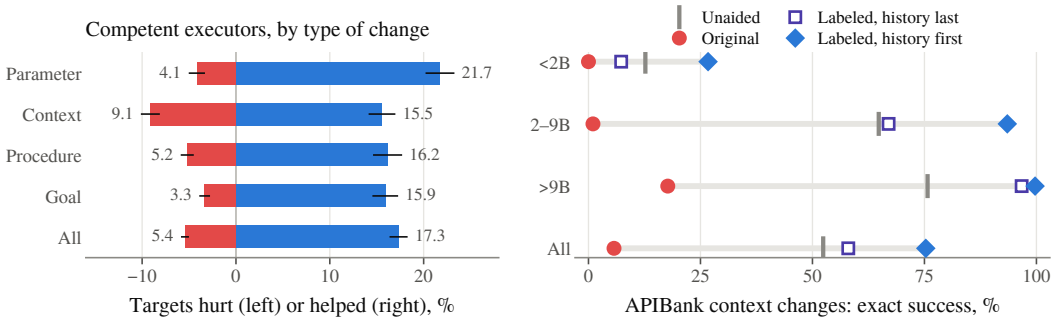

Figure 3: Where a trace helps or hurts. Left: share of targets that the trace turns from failure into success (helped) or from success into failure (hurt), for competent executors, with 95% intervals; the remaining targets are unchanged. Right: success of all 20 checkpoints on APIBank's context-change targets without a trace and under three presentations of the same trace; headline estimates use the history-first presentation.

Key Observation #3

When the interface changes, a trace helps the least, and can even mislead the executor.

4.4 Implications

In this section, we highlight what we believe are the main implications of our findings.

Traces substitute for parameters, above a threshold. For repetitive workloads, a stronger model's trace is worth as much to a competent small executor as doubling its size. Since competence can be measured before any trace is used, a system that pairs a small model with a stronger one can route requests accordingly: requests in environments where the small model has never succeeded go to the stronger model, while elsewhere the small model can reuse the stronger model's traces.

Unaided evaluations underestimate deployed agents. A small model deployed alongside a stronger model's traces can perform like a model twice its size, so evaluating it in isolation underestimates what the deployed system can do, and what it can do wrong. Moreover, since pooling hides how the ability to use a trace grows with scale, evaluations of trace use should compare executors within an environment.

The bottleneck is using a trace, not finding one. Our targets are paired with their sources by construction, i.e., retrieval always returns the right trace. Nevertheless, competent executors close only about a quarter of the gap to the author, and a changed interface can make traces counterproductive. We thus argue that the bottleneck of experience reuse lies in adapting a trace rather than retrieving it, which makes trace use a natural training target.

4.5 Limitations

We note that all traces come from a single author at one reasoning setting, and every source had to be solvable by it: we therefore measure how diverse executors use one strong author's traces. Sources may appear in pretraining data: a sampled memorization diagnostic on all 20 checkpoints found a single weak signal, but cannot rule this out, and contamination would affect both conditions of a pair (Appendix J). Our estimates are specific to one attempt per cell and to our delivery policy; the revised APIBank presentation was adopted after observing the results, without re-rendering its untraced arm. Competence is measured with each benchmark's strict native scoring, so for example in Mind2Web it partly reflects format compliance, and its threshold is an analysis choice. Scale trends rest on four to six checkpoints per family within an environment, and parameter-equivalence is a within-family performance equivalence. Finally, the targets were written by the model that

authored the traces, which may favor its own success on them, making recovery rates conservative (Appendix I), and may align their phrasing with the traces.

5 Related Work

Memory and experience reuse in agents. Reusing a previous solution is a longstanding idea, from case-based reasoning (Aamodt & Plaza, 1994) and derivational analogy (Carbonell & Veloso, 1988) to memory systems for language agents that store feedback, episodes, insights, workflows, skills, or playbooks (Shinn et al., 2023; Zhao et al., 2024a; Wang et al., 2025b; 2023; Ouyang et al., 2025; Zhang et al., 2025a), and to benchmarks that evaluate reuse over task streams or controlled task relationships (Wei et al., 2025; Shu et al., 2026; Belikova et al., 2026; Gao et al., 2026). These works jointly vary what is stored, retrieved, and represented, or measure accumulation over many tasks. We instead fix one verified episode and pair it with its targets by construction, so that our estimates concern how well an executor uses a trace once the right one has been retrieved.

Trajectories as in-context demonstrations. Contextual examples improve language models without parameter updates (Brown et al., 2020; Wei et al., 2022b), and agent methods retrieve, align, or select trajectory demonstrations (Zheng et al., 2024; Zhou et al., 2024; Wang et al., 2025a). Closest to our setting, Gupta et al. (2025) show on AppWorld that trajectories from a stronger model can improve a smaller one; we extend this from one author–executor pair to 20 executors and six environments, with specified changes. Since demonstrations also convey format (Min et al., 2022) and depend on selection and ordering (Liu et al., 2022; Lu et al., 2022), we interpret our gains as the effect of access to a trace under our delivery, and separate the case in which a trace mostly supplies the answer format.

Harmful reuse and scale-dependent use of context. Agents follow retrieved experience even when it no longer applies (Xiong et al., 2026; Hu et al., 2026; Zhan et al., 2026), and context use is sensitive to ordering and position (Shi et al., 2023; Liu et al., 2024). Because our traces are verified and our changes specified, we can distinguish a trace that was wrong from one that no longer applies. Larger models also use context differently from smaller ones (Wei et al., 2023b; Shi et al., 2024; Wei et al., 2023a); we extend this observation to agentic execution, where the share of a trace's value that an executor recovers grows with its scale within an environment.

Expressing external resources in parameters. Retrieval-augmented models treat external information as a complement to parameters (Lewis et al., 2020; Borgeaud et al., 2022), and Le Scao & Rush (2021) express the value of a prompt in equivalent training examples. We follow the latter's logic, but express the value of one trace in model size within a family (Kaplan et al., 2020; Hoffmann et al., 2022), and normalize the gain by the gap to the author, whereas headroom-normalized gains use the maximum possible reward (Asawa et al., 2026). Appendix Q extends this discussion.

6 Conclusion

In this paper, we measured how many parameters a verified execution trace is worth. We showed that, for competent executors, a stronger model's trace of a related task is worth roughly doubling the executor's size, whereas below the competence threshold it is worth almost nothing, unless the executor's failure is due to its answer format. Additionally, we demonstrated that larger executors borrow a larger share of the author's advantage within an environment, and that traces can mislead when the interface changes.

As more of the work of deployed agents shifts to small models, we hope that our protocol and tasks will help extend these measurements to other trace authors, representations, and retrieval methods. But for current models, the answer is clear: a trace can make a small agent act like a larger one, but it cannot teach an agent to act.

AI Use Statement

In this work, we used generative AI tools to help develop the conceptual framework, refine our research questions, provide feedback on the experimental design, implement the experimental and

analysis pipeline, and interpret results. We also used them to generate synthetic data: coding agents running GPT-5.6 Sol, the same model that authored the traces, authored and reviewed the 1,200 targets (D1–D4), together with their fixtures, oracles, checkers, and partial scorers, and in auditing the source tasks and repairing their checkers. Translation and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to identify and summarize related literature, brainstorm framings and titles, suggest a structure for the paper, draft and edit parts of the text, format references, and create the figures and their plotting code. The tools we used were Claude (Fable 5.1 and Opus 5.5), ChatGPT (Sol 5.6 and Astra), and Codex (Sol 5.6 and Astra).

We have reviewed all AI-assisted work. The research question, the notion of borrowed capability, and the study design originated with the authors, who directed every stage of the work through repeated rounds of proposal, critique, and revision, and who rewrote and edited every section of the paper; no AI-generated text was included without review. The authors fixed the scope of the tasks, their semantic policies, and the release criteria. Every number reported in the paper was checked against the outputs of the analysis pipeline, which recomputes each estimate from the content-hashed execution records, and every cited work was checked against its source. Note that LLMs are also the object of our study: the 20 executors are experimental subjects, and the trace author (GPT-5.6 Sol) is both an experimental subject and, as noted above, the model that wrote the targets; their use is described in Section 3 and Appendix F. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Agnar Aamodt and Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1):39–59, 1994. doi: 10.3233/AIC-1994-7104. URL https://journals.sagepub.com/doi/10.3233/AIC-1994-7104.

Parth Asawa, Christopher M. Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E. Gonzalez. Continual Learning Bench: Evaluating Frontier AI Systems in Real-World Stateful Environments, 2026. URL https://arxiv.org/abs/2606.05661.

Marthe Ballon, Brecht Verbeken, Vincent Ginis, and Andres Algaba. Probing the Trajectories of Reasoning Traces in Large Language Models, 2026. URL https://arxiv.org/abs/2601.23163.

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, 2025. URL https://arxiv.org/abs/2506.02153.

Julia Belikova, Rauf Parchiev, Evgeny Egorov, Grigorii Davydenko, Gleb Gusev, Andrey Savchenko, and Maksim Makarenko. Managing Procedural Memory in LLM Agents: Control, Adaptation, and Evaluation, 2026. URL https://arxiv.org/abs/2606.23127.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. Improving Language Models by Retrieving from Trillions of Tokens. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 2022. URL https://proceedings.mlr.press/v162/borgeaud22a.html.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, et al. Language Models are Few-Shot Learners, 2020. URL https://arxiv.org/abs/2005.14165.

Jaime Carbonell and Manuela Veloso. Integrating Derivational Analogy into a General Problem Solving Architecture. Technical report, Carnegie Mellon University, March 1988. URL https://www.cs.cmu.edu/~mmv/papers/CarVel88.pdf.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance, 2023. URL https://arxiv.org/abs/2305.05176.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a Generalist Agent for the Web, 2023. URL https://arxiv.org/abs/2306.06070.

Xingze Gao, Chuanrui Hu, Hongda Chen, Pengfei Yao, Zhao Wang, Yi Bai, Zhengwei Wu, Yunyun Han, Xiaofeng Cong, Jie Gui, Yafeng Deng, and Teng Li. EvoAgentBench: Benchmarking Agent Self-Evolution via Ability Transfer, 2026. URL https://arxiv.org/abs/2607.05202.

Gemini Team. Gemini: A Family of Highly Capable Multimodal Models, 2023. URL https://arxiv.org/abs/2312.11805.

Gemma Team. Gemma 3 Technical Report, 2025. URL https://arxiv.org/abs/2503.19786.

Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, et al. Apple Intelligence Foundation Language Models, 2024. URL https://arxiv.org/abs/2407.21075.

Shivanshu Gupta, Sameer Singh, Ashish Sabharwal, Tushar Khot, and Ben Bogin. Leveraging In-Context Learning for Language Model Agents, 2025. URL https://arxiv.org/abs/2506.13109.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network, 2015. URL https://arxiv.org/abs/1503.02531.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training Compute-Optimal Large Language Models, 2022. URL https://arxiv.org/abs/2203.15556.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8003–8017. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-acl.507. URL https://aclanthology.org/2023.findings-acl.507/.

Qisheng Hu, Quanyu Long, and Wenya Wang. When Continual Learning Moves to Memory: A Study of Experience Reuse in LLM Agents, 2026. URL https://arxiv.org/abs/2604.27003.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models, 2020. URL https://arxiv.org/abs/2001.08361.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, Maxime Gazeau, Himanshu Sahni, Satinder Singh, and Volodymyr Mnih. In-context Reinforcement Learning with Algorithm Distillation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=hy0a5MMPUv.

Teven Le Scao and Alexander M. Rush. How many data points is a prompt worth? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2627–2636. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.naacl-main.208. URL https://aclanthology.org/2021.naacl-main.208/.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, 2020. URL https://arxiv.org/abs/2005.11401.

Laizhen Li, Jiarui Li, Juanjuan Zhao, Kejiang Ye, Ye Li, Cheng-zhong Xu, and Xitong Gao. Grow the Harness, Not the Context: From Strategy-Free Scaffolds to Reusable Specialist Agents, 2026. URL https://arxiv.org/abs/2609.26760.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.187. URL https://aclanthology.org/2023.emnlp-main.187/.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, William B. Dolan, Lawrence Carin, and Weizhu Chen. What Makes Good In-Context Examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pp. 100–114. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.deelio-1.10. URL https://aclanthology.org/2022.deelio-1.10/.

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. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as Agents, 2023. URL https://arxiv.org/abs/2308.03688.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8086–8098. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-long.556. URL https://aclanthology.org/2022.acl-long.556/.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. Teaching Small Language Models to Reason. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1773–1781. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-short.151. URL https://aclanthology.org/2023.acl-short.151/.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. CLIN: A Continually Learning Language Agent for Rapid Task Adaptation and Generalization, 2023. URL https://arxiv.org/abs/2310.10134.

Microsoft. Phi Silica, Small but Mighty On-Device SLM, 2024. URL https://blogs.windows.com/windowsexperience/2024/12/06/phi-silica-small-but-mighty-on-device-slm/. Windows Experience Blog, 6 December 2024. Accessed 2026-09-24.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11048–11064. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-main.759. URL https://aclanthology.org/2022.emnlp-main.759/.

Avanika Narayan, Dan Biderman, Sabri Eyuboglu, Avner May, Scott Linderman, James Zou, and Christopher Re. Cost-efficient Collaboration between On-device and Cloud Language Models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 45682–45719. PMLR, 2025. URL https://proceedings.mlr.press/v267/narayan25a.html.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to Route LLMs with Preference Data, 2024. URL https://arxiv.org/abs/2406.18665.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory, 2025. URL https://arxiv.org/abs/2509.25140.

Bhrij Patel, Ashish Jagmohan, and Aditya Vempaty. Learning API Functionality from In-Context Demonstrations for Tool-based Agents. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 18318–18336. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.994. URL https://aclanthology.org/2025.findings-emnlp.994/.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Qwen Team. Qwen3.5: Towards Native Multimodal Agents, 2026. URL https://qwen.ai/blog?id=qwen3.5. Official model-family announcement. Accessed 2026-09-24.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4902–4912. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.442. URL https://aclanthology.org/2020.acl-main.442/.

Yangjun Ruan, Chris J. Maddison, and Tatsunori Hashimoto. Observational Scaling Laws and the Predictability of Language Model Performance. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.10938.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. Learning To Retrieve Prompts for In-Context Learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2655–2671. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.naacl-main.191. URL https://aclanthology.org/2022.naacl-main.191/.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. Large Language Models Can Be Easily Distracted by Irrelevant Context, 2023. URL https://arxiv.org/abs/2302.00093.

Zhenmei Shi, Junyi Wei, Zhuoyan Xu, and Yingyu Liang. Why Larger Language Models Do In-context Learning Differently? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 44991–45013. PMLR, 2024. URL https://proceedings.mlr.press/v235/shi24f.html.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning, 2023. URL https://arxiv.org/abs/2303.11366.

Yiheng Shu, Bernal Jiménez Gutiérrez, Saisri Padmaja Jonnalagedda, Yuguang Yao, Huan Sun, and Yu Su. AgentCL: Toward Rigorous Evaluation of Continual Learning in Language Agents, 2026. URL https://arxiv.org/abs/2606.02461.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic Cheatsheet: Test-Time Learning with Adaptive Memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-long.333. URL https://aclanthology.org/2026.eacl-long.333/.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models, 2023. URL https://arxiv.org/abs/2305.16291.

Ruoyu Wang, Junda Wu, Yu Xia, Tong Yu, Ryan A. Rossi, Julian McAuley, and Lina Yao. DICE: Dynamic In-Context Example Selection in LLM Agents via Efficient Knowledge Transfer, 2025a. URL https://arxiv.org/abs/2507.23554.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent Workflow Memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research*, 2022a. URL https://arxiv.org/abs/2206.07682.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, 2022b. URL https://arxiv.org/abs/2201.11903.

Jerry Wei, Le Hou, Andrew Lampinen, Xiangning Chen, Da Huang, Yi Tay, Xinyun Chen, Yifeng Lu, Denny Zhou, Tengyu Ma, and Quoc Le. Symbol tuning improves in-context learning in language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 968–979. Association for Computational Linguistics, 2023a. doi: 10.18653/v1/2023.emnlp-main.61. URL https://aclanthology.org/2023.emnlp-main.61/.

Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, and Tengyu Ma. Larger language models do in-context learning differently, 2023b. URL https://arxiv.org/abs/2303.03846.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-Memory: Benchmarking LLM Agent Test-time Learning with Self-Evolving Memory, 2025. URL https://arxiv.org/abs/2511.20857.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How Memory Management Impacts LLM Agents: An Empirical Study of Experience-Following Behavior. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 623–645. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.27. URL https://aclanthology.org/2026.acl-long.27/.

Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua Tenenbaum, and Chuang Gan. Prompting Decision Transformer for Few-Shot Policy Generalization. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine*

Learning Research, pp. 24631–24645. PMLR, 2022. URL https://proceedings.mlr.press/v162/xu22g.html.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen3 Technical Report, 2025. URL https://arxiv.org/abs/2505.09388.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and Benchmarking Interactive Coding with Execution Feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.nips.cc/paper/2023/hash/4b175d846fb008d540d233c188379ff9-Abstract-Datasets_and_Benchmarks.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models, 2022. URL https://arxiv.org/abs/2210.03629.

Michihiro Yasunaga, Xinyun Chen, Yujia Li, Panupong Pasupat, Jure Leskovec, Percy Liang, Ed H. Chi, and Denny Zhou. Large Language Models as Analogical Reasoners, 2023. URL https://arxiv.org/abs/2310.01714.

Yuhao Zhan, Bingxiang He, Zecong Tang, and Chaojun Xiao. PACE-Bench: Benchmarking Physics Adaptation via Code Evolution in Dynamic Environments, 2026. URL https://arxiv.org/abs/2608.14441.

Danyang Zhang, Lu Chen, Situo Zhang, Hongshen Xu, Zihan Zhao, and Kai Yu. Large Language Models Are Semi-Parametric Reinforcement Learning Agents, 2023. URL https://arxiv.org/abs/2306.07929.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models, 2025a. URL https://arxiv.org/abs/2510.04618.

Qizheng Zhang, Michael Wornow, Gerry Wan, and Kunle Olukotun. Agentic Plan Caching: Test-Time Memory for Fast and Cost-Efficient LLM Agents, 2025b. URL https://arxiv.org/abs/2506.14852.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM Agents Are Experiential Learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642, 2024a. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Jiachen Zhao, Zonghai Yao, Zhichao Yang, and Hong Yu. Large Language Models are In-context Teachers for Knowledge Reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 16470–16486. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.findings-emnlp.961. URL https://aclanthology.org/2024.findings-emnlp.961/.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-Exemplar Prompting with Memory for Computer Control. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Pc8AU1aF5e.

Ruiwen Zhou, Yingxuan Yang, Muning Wen, Ying Wen, Wenhao Wang, Chunling Xi, Guoqiang Xu, Yong Yu, and Weinan Zhang. TRAD: Enhancing LLM Agents with Step-Wise Thought Retrieval and Aligned Decision, 2024. URL https://arxiv.org/abs/2403.06221.

Jingwei Zuo, Maksim Velikanov, Ilyas Chahed, Younes Belkada, Dhia Eddine Rhayem, Guillaume Kunsch, Hakim Hacid, Hamza Yous, Brahim Farhat, Ibrahim Khadraoui, Mugariya Farooq, Giulia Campesan, Ruxandra Cojocaru, Yasser Djilali, Shi Hu, Iheb Chaabane, Puneesh Khanna, Mohamed El Amine Seddik, Ngoc Dung Huynh, Phuc Le Khac, Leen AlQadi, Billel Mokeddem, Mohamed Chami, Abdalgader Abubaker, Mikhail Lubinets, Kacper Piskorski, and Slim Frikha. Falcon-H1: A Family of Hybrid-Head Language Models Redefining Efficiency and Performance, 2025. URL https://arxiv.org/abs/2507.22448.

A Source Audit, Selection, and Trace Eligibility

Auditing 4,221 candidates. The source-selection process audited 4,221 items across the six source populations (Table 2). Tasks and checkers were assessed on separate axes. On the task axis, an item is *valid*, has *acceptable issues* (i.e., only an overzealous reading finds a problem, and the issue cannot meaningfully change execution or evaluation), or has *unacceptable issues* (i.e., missing, conflicting, or unrecoverable information can materially affect execution or scoring). An item with a critical task defect was excluded even if its checker could be made to pass. Conversely, a sound task with a checker defect remained eligible only when a repair could be derived from the public request, the visible state, and the upstream semantics without inventing new gold solutions.

Distribution-matched selection. Selection approximated the upstream distribution marginals of each benchmark subject to eligibility, repairability, and caps on repeated tasks, using deterministic optimization with stable hash tie-breaking. Responses, pass rates, and perceived difficulty for the evaluated open models were unavailable to the selector. The initial 300 sources contained 286 valid items and 14 items with acceptable issues; 246 required some repair of the exact checker under our scoring contract.

Repairing overly rigid checkers. All selected sources were reviewed for excessive rigidity, i.e., whether a meaningfully correct execution could be rejected because the checker enforced an unspecified ordering, an incidental representation, or only one of several native-equivalent answers. Repairs were narrow and evidence-based (e.g., accepting either projection order in SQL when the request does not specify one); broad fuzzy matching was not introduced.

One verified trace per source. Every source must have an exact, verified author trace under the frozen request, harness, and checker. The author was allowed at most five whole-task attempts; operational failures did not consume an attempt, while a valid scored failure did. The first campaign produced 275 exact traces and 25 holds. For a held source, independently justified checker or harness defects were first repaired; if the item remained sound but still lacked an exact trace, it was replaced according to a candidate order fixed in advance by task readiness, distribution match, presentation clarity, and a stable sampling hash. Open-model results were forbidden inputs at every step. The final release contains 12 such replacements.

Table 2: Source selection and replacement accounting.

Benchmark	Audited	Selected	Replaced	Final
AppWorld	732	50	0	50
AgentBench OS	144	50	0	50
InterCode SQL	1,034	50	3	50
BFCL	800	50	0	50
APIBank	172	50	1	50
Mind2Web	1,339	50	8	50
Total	4,221	300	12	300

B Target Types: Rubric and Released Examples

Distinguishing the four types. The four adaptation tasks change relative to the source. D1 changes the arguments of substantially the same solution; D2 changes the mapping from that solution to the environment; D3 changes the structure of the solution; and D4 changes the requested outcome while retaining concrete source knowledge. Arbitrary tags, confirmation codes, and decorative fields do not count as D1. For D2, gratuitous wrappers or renames created only to defeat replay are disallowed. For D3, the changed step must contribute to the requested outcome: decorative validation, duplicate work, or multiplying the task solely to manufacture an extra step is forbidden. For D4, the source outcome must not satisfy the target, and a different output wrapper or a reformatted source answer is not an adjacent goal.

APIBank context changes. The D2 targets of APIBank are a particularly controlled instance of context change: the operation, the dialogue, the values, and the one-call procedure of the source are preserved, while exactly one active and meaningful parameter name changes. The target prompt states that the current API description is authoritative and that parameter names from the source interface are unavailable, but it reveals neither which field changed nor the mapping.

Matching difficulty to the source. Our review compared substantive reasoning and interaction steps, tool and environment breadth, grounding and search burden, state tracking, expected solution length, ambiguity, and error recovery. The only passing judgment is *roughly the same*; a materially easier or harder target must be revised or replaced. This measure is inherently fuzzy, since using universal measures (e.g. number of steps) is not possible across our breadth of tasks. We stress that the author's success on a target is a post-review feasibility diagnostic and cannot substitute for this judgment.

Released examples. Table 3 reports three released source families. They are taken from the accepted targets and are not templates from which other targets were generated.

Table 3: Three released source–target families (requests paraphrased).

	AppWorld	AgentBench OS	InterCode SQL
Source	Mark as read every inbox and outbox thread from before the last calendar month	Count the regular files and directories directly under `/etc`	Count the cars with more than six cylinders
D1	Cutoff becomes the current calendar year	Same count under `/var`	More than four cylinders
D2	Same request through a reorganized workspace SDK with renamed operations	Same count from an offline typed inventory file	Same question on a descriptive reporting schema with renamed tables and fields
D3	Mark qualifying inbox threads as read and archive qualifying outbox threads	First retain a sorted typed inventory, then derive the count from it	Counts grouped by cylinder count, for values above six
D4	Archive the qualifying unread threads while preserving read state	Separate counts, their difference, and a retained typed report	Average horsepower of cars above six cylinders

C How Targets Were Authored and Reviewed

Both the authoring and the item-level review of targets were performed by coding agents running GPT-5.6 Sol, the same model that authored the traces; we fixed the rubric, the review gates, and the release criteria, but did not review items systematically by hand. For every candidate target, the authoring record identifies: the complete target request and its semantic difference from the source; the claimed type and the nearest competing type; solution sketches for source and target; the trace facts or operations expected to transfer; what must change rather than be copied; an intrinsic-difficulty comparison; the target fixture, oracle, exact checker, and partial scorer; and source-replay and targeted near-miss controls.

Each accepted target then received an item-level review by an agent in a separate turn, which inspects the actual artifacts rather than a generated summary and writes a rationale for each gate: naturalness; material change; correct type; concrete trace utility; adaptation rather than replay or restart; native feasibility; roughly equal intrinsic difficulty; exact-checker correctness; meaningful partial credit; leakage; and outcome-independent authoring. Each acceptance binds the reviewed request, fixture, oracle, checker, partial scorer, source trace, runtime evidence, reviewer identity, timestamp, repository commit, and content hashes. Any material revision invalidates the acceptance, and the release contains no acceptances inherited from the withdrawn programmatic targets.

Failures of the author were allowed to identify locations for inspection, but a repair required independent support from the public request, the visible interface, the upstream semantics, or a reproducible harness defect. The release is content-addressed, and all release builders fail closed on missing, duplicate, stale, or hash-drifted inputs. Release gates established: 300 sources and 1,200 unique

targets, balanced by benchmark and type; 1,200 accepted reviews; all 600 static oracles exact and all 600 blank controls at zero; executable qualification for all 1,200 targets, with every transformation type exercised end-to-end in all three stateful routes; and first-turn context qualification for all 1,200 targets in both conditions, with no overflow.

D Exact and Partial Scoring per Route

Exact success requires every material public requirement. Partial scores reward substantive progress without weakening the exact endpoint, and are binary when no reasonable intermediate state exists: among the 600 static targets, 526 admit nontrivial fractional progress, while 74 contain a single material component and are thus retained as binary. For every route, a blank or no-op control scores zero, and checkers accept independently justified equivalent solutions but grant no credit for generic activity, protocol compliance, or invariants that were already true before the model acts. Table 4 summarizes the route-specific scorers. Static routes follow each benchmark's native output contract. For example, the Mind2Web prompt asks for exactly one JSON object with three keys, and the scorer parses the entire completion, so a correct object inside a Markdown code block fails. We did not relax these parsers; Appendix M reports how often executors fail them.

Table 4: Route-specific execution and scoring.

Route	Interaction	Exact scoring	Partial scoring
AppWorld	APIs over fresh application state	Native state tests, including preservation requirements	Goal and preservation requirement families
AgentBench OS	Serial shell actions in an isolated container	Source-derived and target-specific state checks	Meaningful state components; binary when none is defensible
InterCode SQL	Up to 10 textual SQL actions with database feedback	Semantic result comparison with an explicit order policy	Maximum substantive result score reached in the episode
BFCL	Bounded textual typed-call response	Type-specific typed call set, context migration, ordered procedure, or adjacent operation	Required functions and material typed arguments
APIBank	Bounded textual API-call response	Required API and all material typed arguments; an unavailable legacy field cannot receive exact credit	API selection and material argument components
Mind2Web	Bounded grounded action response	Grounded element, operation, and value	Element, operation, and value, scored separately

E How Traces Are Represented and Delivered

Trace schema. Traces preserve the order of observable events. Author actions are retained byte-for-byte; tool and environment events retain role, ordinal, content, and provenance. Across the 300 sources, the full observable events amount to 3.66MB.

Inline and projected delivery. BFCL, APIBank, Mind2Web, AgentBench OS, and InterCode SQL receive the complete observable trace inline, with no retrieval tool in either condition. For AppWorld, every projected event includes a descriptor with its exact byte length, hash, format class, and retrieval key. Small observations are included verbatim; large structured observations retain a deterministic schema, the records referenced by later author actions, and bounded samples, within a cap of 20,000 rendered bytes per event. The projection is computed from the source only. The omitted bytes remain available through two tools: an exact read by event and offset, and a case-insensitive literal search with bounded context. Retrieval is optional and interleaved with ordinary actions; there is no forced retrieval phase. The two tools share 16 attempted calls, 4,096 bytes per response, and 65,536 response bytes overall, and invalid attempts count against the allowance. The NoTrace condition never receives a retrieval tool.

Prompts per condition. The schematic below is explanatory and does not replace the exact serialized presentations in the release.

```
TRACE       system: native route instructions, including the current
                    interface description, + trace-use instructions
            user:   source task as originally presented (including its
                    interface description) + verified observable trace,
                    followed by the target request
            tools:  native tools (+ AppWorld trace read/search)

NOTRACE     system/user: native target presentation
            tools:       native tools only
```

In TRACE, the source task and its trace therefore follow the system instructions. For APIBank and BFCL, these instructions contain the complete current interface, and the final user message adds only the current request, so that on context changes the last interface the executor reads is the obsolete one. InterCode SQL differs: the current schema is never printed, and the executor must discover it by querying the database. For APIBank's context-change targets, all headline estimates instead use the revised presentation of Appendix O:

```
TRACE       system: fixed wrapper
(revised)   user:   [HISTORICAL] source task as originally presented +
                    verified observable trace
                    [CURRENT] complete current task, including its
                    interface description and dialogue
            tools:  native tools
```

F CHECKPOINTS AND SERVING ARTIFACTS

Reasoning. Qwen3 and Qwen3.5 executors run with thinking enabled, i.e., without any request that disables it, and their reasoning is separated from the scored output by the serving stack's Qwen3 reasoning parser; the saved qualification responses confirm that these checkpoints produce reasoning. Decoding is greedy in both conditions, which differs from the sampling settings recommended for Qwen's thinking mode and may lower unaided and traced success alike. The memorization probes of Appendix J, by contrast, disable thinking where the chat template supports it.

The benchmark worker and the model server are separate. A qualified endpoint must provide OpenAI-compatible chat completions and a live tokenization endpoint that uses the same template as generation. An endpoint receipt binds the checkpoint revision and snapshot digest, the precision or quantized artifact, the serving implementation and its image or executable digest, the accelerator and architecture, the context length, the chat template, the tool-call and reasoning parsers, and the generation policy. Before bulk collection, an endpoint must pass identity, tokenization, plain-text, forced-tool, multi-turn tool-return, and route canaries in both conditions. Reference implementations include vLLM on CUDA, an exact BF16 server on Apple MPS, and a gateway around a pinned llama.cpp binary for pinned GGUF artifacts. Both conditions of a checkpoint share one endpoint identity; hardware may vary across checkpoints only as recorded metadata, and we make no claim of bitwise equivalence across substrates. Hosted providers that do not expose exact local rendering control are excluded. Static routes do not become native-tool tasks at the serving boundary, and SQL remains an iterative textual loop. Table 5 reports the substrate and artifact of every checkpoint.

Falcon-H1 34B artifact deviation. The frozen matrix pins a 4-bit GGUF quantization (Q4_K_M) for Falcon-H1 34B. The executed artifact was the 8-bit Q8_0 quantization of the same checkpoint revision on BFCL, APIBank, Mind2Web, and InterCode SQL, and the pinned Q4_K_M on AppWorld and AgentBench OS, as attested by the execution records; the two labeled APIBank presentations of Appendix O, including the one used in headline estimates, also used Q4_K_M. Of the checkpoint's 2,400 analyzed cells, 1,550 were therefore served at Q8_0 and 850 at Q4_K_M, and its 50 revised APIBank context-change pairs compare a Q4_K_M traced arm with a Q8_0 untraced arm. We report the checkpoint as it was run, and Table 8 gives the headline estimates without it.

Table 5: Frozen checkpoint identities. Revisions are immutable Hugging Face commits, abbreviated to 12 characters; full hashes, serving substrates, templates, and parsers are bound in the released endpoint receipts. "HF (vLLM)" denotes the Hugging Face snapshot served with vLLM; the receipts attest the artifact but not its numerical precision. †The pinned artifact is Q4_K_M; see the artifact deviation above.

#	Family	Size	Repository	Revision	Serving artifact
1	Qwen3	0.6B	`Qwen/Qwen3-0.6B`	`c1899de289a0`	GGUF Q8_0
2	Qwen3	1.7B	`Qwen/Qwen3-1.7B`	`70d244cc86cc`	HF (vLLM)
3	Qwen3	4B	`Qwen/Qwen3-4B`	`1cfa9a720891`	HF (vLLM)
4	Qwen3	8B	`Qwen/Qwen3-8B`	`b968826d9c46`	HF (vLLM)
5	Qwen3	14B	`Qwen/Qwen3-14B`	`40c069824f42`	HF (vLLM)
6	Qwen3	32B	`Qwen/Qwen3-32B`	`9216db5781bf`	HF (vLLM)
7	Qwen3.5	0.8B	`Qwen/Qwen3.5-0.8B`	`2fc06364715b`	HF (vLLM)
8	Qwen3.5	2B	`Qwen/Qwen3.5-2B`	`15852e8c1636`	HF (vLLM)
9	Qwen3.5	4B	`Qwen/Qwen3.5-4B`	`851bf6e806ef`	HF (vLLM)
10	Qwen3.5	9B	`Qwen/Qwen3.5-9B`	`c20223623576`	HF (vLLM)
11	Qwen3.5	27B	`Qwen/Qwen3.5-27B`	`fc05daec18b0`	HF (vLLM)
12	Gemma 3	270M	`google/gemma-3-270m-it`	`ac82b4e82054`	GGUF Q8_0
13	Gemma 3	1B	`google/gemma-3-1b-it`	`dcc83ea841ab`	GGUF Q8_0
14	Gemma 3	4B	`google/gemma-3-4b-it`	`093f9f388b31`	HF (vLLM)
15	Gemma 3	12B	`google/gemma-3-12b-it`	`96b6f1eccf38`	HF (vLLM)
16	Gemma 3	27B	`google/gemma-3-27b-it`	`005ad3404e59`	HF (vLLM)
17	Falcon-H1	1.5B	`tiiuae/Falcon-H1-1.5B-Instruct`	`80ebc50d7799`	GGUF Q8_0
18	Falcon-H1	3B	`tiiuae/Falcon-H1-3B-Instruct`	`01087ec4c132`	GGUF Q8_0
19	Falcon-H1	7B	`tiiuae/Falcon-H1-7B-Instruct`	`41e72f27effb`	HF (vLLM)
20	Falcon-H1	34B	`tiiuae/Falcon-H1-34B-Instruct`	`6e6890e26cba`	Q8_0 (static, SQL); Q4_K_M (AppWorld, OS)†

G Budgets, Context Limits, and Failure Handling

Interaction budgets. Table 6 reports the interaction limits. Reaching a turn, output, action, or context limit yields an explicit terminal reason and a score for the state reached.

Table 6: Interaction limits. Output limits are in tokens; retained environment output is in bytes.

Route	Turns	Per turn	Whole task	Env. bytes	Native cap
BFCL / APIBank / Mind2Web	4	4,096	8,192	4,096	–
AgentBench OS	32	2,048	16,384	32,768	route termination
InterCode SQL	12	4,096	16,384	32,768	10 SQL actions
AppWorld	64	2,048	16,384	32,768	route termination

Context management. Before every generation, the serving endpoint tokenizes the exact rendered messages and tool schemas. If the prompt plus the next output allowance would overflow, only old environment output is replaced with bounded content-addressed markers, and the prompt is checked again; author actions, target requests, and model outputs are never rewritten. A request that still does not fit is an operational no-score. An offline audit of all 2,400 first-turn envelopes per tokenizer family found no overflow.

Error feedback. Malformed JSON, unknown tools, invalid trace lookups, and multiple calls under the serial contract are preserved as model output and are not executed. The executor receives a bounded error that names the attempted tool, the valid tools, the expected arguments, and the recovery rule, without any task-specific advice. Multiple calls are rejected atomically rather than partially executed.

State isolation. Every attempt starts from fresh state: AppWorld task state, a fresh MySQL instance for InterCode SQL, and a hash-qualified container for AgentBench OS, where concurrent workers receive isolated network lanes. State is never shared between models, targets, or conditions, and cleanup is verified against exact resource identities.

Failure attribution and retries. Terminal outcomes are classified as exact success, scored model failure, provider or transport failure, harness or environment failure, evaluator failure, or unsupported checkpoint. Only the first two enter performance denominators. An operational rerun uses a

content-bound successor plan that preserves the earlier evidence; no completed scored cell is ever redispatched, and duplicate or conflicting terminals fail reconciliation.

Collection order by scale. Collection is staged by advertised scale: each tier is released only once all smaller tiers have complete scored coverage in both conditions. This ordering changes resource allocation, not the frozen task matrix or the one-attempt policy.

H Bootstrap, Multipliers, and Scale Trends

Complete coverage. The final analysis has a score for every one of the 48,000 target cells and the 6,000 source cells, so every estimate uses complete pairs and no source-competence stratum is unknown. Operational failures that were rerun under a successor plan retain their original evidence (Appendix G).

Environment-stratified bootstrap. For each of 10,000 replicates (seed 20260925), we sample, within each environment, 50 source identifiers with replacement, and carry along all targets and both conditions of each sampled source; the same draw is shared across checkpoints, transformation types, and conditions. Means, differences, recovery rates, slopes, and multipliers are recomputed on each replicate. Stratifying by environment holds the benchmark weights fixed, and sharing draws across checkpoints avoids treating several checkpoints on the same source as independent. Pooled estimates weight each environment by its number of competent blocks. Membership in the set of competent blocks is held fixed, so intervals are conditional on the observed qualification. Intervals are pointwise and describe variation over source tasks for this fixed set of checkpoints and single attempts; they do not include generation randomness or uncertainty over a population of models.

Competence threshold. An executor is competent in an environment if it reaches exact success on at least k of the environment's 50 sources in the separate no-trace run, with one attempt per checkpoint and source; $k = 1$ is primary and $k = 5$ a sensitivity analysis. Every target of a qualifying checkpoint–environment block is eligible, including targets whose own source was not solved. At $k = 1$, 89 of the 120 blocks qualify, and 75 at $k = 5$. Of the 31 blocks below the threshold at $k = 1$, 13 are in AppWorld, 7 in AgentBench OS, 9 in Mind2Web (including the six format-only blocks), and one each in BFCL and APIBank. The within-target split by source success (Appendix L) uses the same run; it is descriptive, since sources that an executor solves differ in difficulty and headroom from those it does not.

Parameter multipliers. For each family, we regress mean exact success on log parameter count and trace access, which imposes parallel traced and untraced curves; the implied multiplier is $\exp(\gamma/\beta)$, the exponential of the ratio of the trace coefficient to the size coefficient. The primary fit uses the mean of each qualifying checkpoint–environment block, with equal weight per block and environment intercepts; the pooled fit additionally has family intercepts and a shared slope and trace coefficient. As a robustness check, we fit checkpoint means over each checkpoint's qualifying blocks without environment intercepts, which gives 2.6 pooled (Table 11). A multiplier is not reported if fewer than 95% of bootstrap size slopes are positive. Both fits are descriptive: the curves are not always parallel, and many implied sizes lie outside the observed range of a family.

Within-environment scale trends. For each family and environment, we regress the recovery rate of each qualifying checkpoint on $\log_2$ of its parameter count, with equal weight per checkpoint, so that slopes are in units of recovery per doubling. Recovery pools the retained pairs of parameter, procedure, and goal changes before dividing. A slope is not reported when only one size qualifies or when any qualifying checkpoint's gap to the author is below 0.05; we never drop such a checkpoint to obtain a slope. Pooled across environments, the same regression controls the family but not the changing mix of environments.

Excluding format failures. A pair is excluded in the format-failure sensitivity analysis if either condition has a positive parse-failure flag from the native scorer. Static executions whose parser status could not be classified are retained, and tool or SQL errors in stateful routes are not treated as format failures. Since the exclusion depends on both outcomes, it removes the cases in which a

trace teaches the format as well as those in which it breaks it, and it changes the composition of each checkpoint's targets; the resulting estimates are descriptive.

I THE AUTHOR'S SUCCESS ON THE TARGETS

The author's feasibility check reached exact success within five attempts on 1,147 of the 1,150 targets it attempted in their current form, with a mean best-attempt partial score of 0.998; the three failures are the D1–D3 targets of a single InterCode SQL source and were independently classified as genuine model errors. The 50 APIBank D2 targets, redesigned after that campaign, were attempted in a separate run by the same author at the same reasoning setting, under that campaign's own budgets, and all 50 succeeded on the first attempt. We use these runs as a solvability diagnostic and as the author reference $\bar{S}_A$ in Eq. 2; they are neither open-model results nor criteria for target selection. Since the targets were written by agents running the same model, $\bar{S}_A$ may overstate how often an author that did not write them would succeed. A higher $\bar{S}_A$ lowers every recovery rate, and lowers it most for the executors closest to the author, so this bias would make both recovery rates and their rise with scale conservative; gains and parameter multipliers do not depend on $\bar{S}_A$. For $\bar{S}_A$ we use the author's first attempt, the quantity comparable to the executors' single attempt, for 1,182 targets, and success within five attempts for the 18 whose first attempt was not scored.

J MEMORIZATION CHECK

We ran a sampled behavioral memorization diagnostic on all 20 checkpoints, as a diagnostic rather than a curation gate. Each checkpoint receives the same 48 probes, frozen during task construction and selected without access to any evaluated model output, for 960 responses overall; the trace author is not probed. Unlike in the main experiments, thinking is disabled where the chat template supports it. Each probe receives a single attempt.

Prefix test. For each of the six benchmarks, two task-text prefixes are each paired with an authored control prefix (24 probes per checkpoint). A probe is positive if the model reproduces the benchmark continuation exactly, without normalizing whitespace or formatting, but not the control continuation. This test detects verbatim familiarity only, and misses paraphrased or partial memorization.

Obsolete-answer test. For BFCL, APIBank, and Mind2Web, two base tasks each have two independently changed variants, and each variant is paired with its own attempt at the original task (24 probes per checkpoint). A variant qualifies if the model solves the original task, fails the changed one, and its answer to the changed task passes the original checker, i.e., it returns the obsolete answer. A base task is flagged only if both of its variants qualify. All conditions use the frozen task checkers rather than string equality.

Results. Table 7 reports the results per checkpoint. We observe a single positive prefix probe, for Qwen3.5 27B on an APIBank example in which the user is asked which text to translate, and the continuation is the pangram "The quick brown fox jumps over the lazy dog." Since the pangram is a common example sentence, we consider this signal weak, but we still report it. No case meets the obsolete-answer criterion, and no base task is flagged. A separate prefix check on three of the replacement sources found no signal in 24 model–item pairs.

Scope. We stress that these diagnostics cannot establish that the sources were absent from training. The sample is small, and the obsolete-answer test is informative only when the model solves the original task, which happens in 124 of the 240 paired cases, and never for Qwen3 0.6B, Qwen3.5 0.8B, or Gemma 3 270M and 1B. AppWorld, AgentBench OS, and InterCode SQL are covered by the prefix test only. Moreover, 14 checkpoints were probed in earlier runs on heterogeneous substrates, including a hosted API and quantized GGUF artifacts, and the other six on their official BF16 weights. We note that contamination would affect both conditions of a pair, and that the targets, unlike the sources, were authored for this work.

Table 7: Memorization diagnostics per checkpoint: positive prefix probes (out of 12 benchmark–control pairs), and original and changed tasks solved in the obsolete-answer test (out of 12 each). No checkpoint returns a qualifying obsolete answer.

Checkpoint	Prefix	Original	Changed	Checkpoint	Prefix	Original	Changed
Qwen3 0.6B	0	0	0	Gemma 3 270M	0	0	0
Qwen3 1.7B	0	8	9	Gemma 3 1B	0	0	0
Qwen3 4B	0	7	5	Gemma 3 4B	0	6	6
Qwen3 8B	0	10	9	Gemma 3 12B	0	8	7
Qwen3 14B	0	12	10	Gemma 3 27B	0	7	6
Qwen3 32B	0	12	8	Falcon-H1 1.5B	0	2	0
Qwen3.5 0.8B	0	0	0	Falcon-H1 3B	0	4	4
Qwen3.5 2B	0	6	8	Falcon-H1 7B	0	8	7
Qwen3.5 4B	0	10	8	Falcon-H1 34B	0	6	4
Qwen3.5 9B	0	8	8				
Qwen3.5 27B	1	10	7				

K Results per Checkpoint, Environment, and Family

Table 8 reports the headline estimates under alternative analysis choices. Table 9 reports, for every checkpoint, the number of environments in which it is competent and its gain there, together with unaided and traced success and the gain and recovery rate on each transformation type over all 1,200 targets. Table 10 reports the gain for every environment, over all blocks and for competent executors, and Table 11 the gain and parameter multipliers for every family. All use the revised APIBank presentation (Section 3). Figures 4 and 5 show the corresponding estimates with their intervals. Partial-score versions and 95% intervals for every row are included in the released analysis files; for competent executors, the partial-score gain is 11.2 points (95% CI 10.2–12.3), and it is positive for every checkpoint, family, and environment.

Table 8: Gain from trace access (points) for competent executors, with the 95% interval for the overall gain and the pooled parameter multiplier, under the primary analysis and alternative choices. "Without format failures" selects pairs by their outcomes and is descriptive. Without environment intercepts, the primary multiplier is 2.6. ‡Checkpoint-mean fit.

Analysis	Pairs	Overall	D1	D2	D3	D4	Multiplier
Primary ($k = 1$)	17,800	11.9 [10.7, 13.1]	17.6	6.4	11.0	12.6	2.05×
Stricter threshold ($k = 5$)	15,000	11.9 [10.6, 13.3]	17.7	6.1	11.7	12.3	2.30×
Without format failures	15,816	9.5 [8.3, 10.8]	14.3	3.7	10.3	9.8	1.85×
Original APIBank presentation	17,800	8.0 [6.8, 9.2]	17.6	−9.2	11.0	12.6	1.64×
Without APIBank D2	16,850	11.2 [10.0, 12.4]	17.6	1.6	11.0	12.6	1.94×
Without Falcon-H1 34B	17,000	11.5 [10.3, 12.7]	17.7	5.4	10.6	12.3	1.93×
All blocks	24,000	12.3 [11.4, 13.2]	16.4	8.2	11.6	13.1	2.50×‡

L Gains on Tasks the Executor Could and Could Not Solve

Table 12 splits each executor's targets, over all blocks, by whether it solved the corresponding source without a trace. On targets from unsolved sources, traced executors succeed far more often than unaided ones, in every size tier and on every transformation type shown. Much of the gain therefore is due to tasks whose source the executor could not solve on its own. It does not, however, favor such tasks relative to the room left for improvement: on parameter changes, it closes 27% of the gap for unsolved sources against 31% for solved ones, and on procedure and goal changes the solved stratum recovers more. The concentration of absolute gains on unsolved sources thus mostly reflects where there is headroom. Context changes are omitted, since their traced arm differs by presentation in APIBank.

Table 9: Per-checkpoint results. "Envs" is the number of environments in which the checkpoint is competent, and "Competent" its gain there (points). The remaining columns cover all 1,200 targets: unaided and traced exact success (%), and the gain (points) with the recovery rate in parentheses on each transformation type. †Served from a non-pinned artifact on most routes (Appendix F).

Checkpoint	Envs	Competent	Unaided	Traced	D1	D2	D3	D4
Qwen3 0.6B	4	+15.8	13.5	24.3	+19.0 (.23)	+1.3 (.02)	+5.0 (.06)	+17.7 (.21)
Qwen3 1.7B	5	+12.4	38.4	48.8	+22.0 (.37)	−7.3 (−.12)	+11.7 (.19)	+15.0 (.26)
Qwen3 4B	6	+4.3	53.9	58.3	+14.7 (.33)	−9.7 (−.21)	+4.7 (.10)	+7.7 (.18)
Qwen3 8B	6	+5.2	58.9	64.1	+11.3 (.32)	−3.7 (−.08)	+9.3 (.21)	+3.7 (.10)
Qwen3 14B	6	+6.8	61.4	68.3	+10.7 (.32)	+2.0 (.05)	+8.7 (.21)	+6.0 (.17)
Qwen3 32B	6	+6.2	64.5	70.7	+11.3 (.38)	−0.3 (−.01)	+10.0 (.25)	+3.7 (.11)
Qwen3.5 0.8B	4	+6.6	7.6	16.8	+16.3 (.18)	+4.7 (.05)	+6.0 (.06)	+10.0 (.11)
Qwen3.5 2B	4	+19.1	21.0	34.3	+12.7 (.17)	+19.3 (.22)	+9.3 (.12)	+11.7 (.16)
Qwen3.5 4B	6	+10.1	58.7	68.8	+13.3 (.33)	+10.3 (.22)	+11.0 (.26)	+5.7 (.18)
Qwen3.5 9B	6	+8.1	67.1	75.2	+10.0 (.37)	+6.0 (.16)	+7.7 (.23)	+8.7 (.31)
Qwen3.5 27B	6	+12.1	67.2	79.3	+9.3 (.34)	+14.7 (.40)	+13.0 (.42)	+11.3 (.36)
Gemma 3 270M	1	+10.0	0.1	1.8	+5.7 (.06)	+0.0 (.00)	+0.0 (.00)	+1.0 (.01)
Gemma 3 1B	3	+13.2	1.8	8.8	+15.7 (.16)	−1.0 (−.01)	+4.0 (.04)	+9.3 (.10)
Gemma 3 4B	4	+8.6	30.8	38.6	+10.0 (.16)	+4.0 (.06)	+9.0 (.12)	+8.0 (.13)
Gemma 3 12B	4	+7.6	41.3	61.0	+20.0 (.37)	+18.7 (.34)	+19.3 (.29)	+21.0 (.38)
Gemma 3 27B	4	+7.9	44.3	63.4	+18.0 (.33)	+20.7 (.39)	+19.0 (.33)	+18.7 (.35)
Falcon-H1 1.5B	4	+31.0	7.3	28.2	+29.7 (.34)	+22.0 (.23)	+10.0 (.11)	+21.7 (.24)
Falcon-H1 3B	3	+32.0	18.0	37.3	+25.3 (.34)	+12.7 (.15)	+14.7 (.18)	+24.3 (.31)
Falcon-H1 7B	3	+21.7	32.1	57.5	+26.0 (.41)	+16.0 (.25)	+30.7 (.41)	+29.0 (.44)
Falcon-H1 34B†	4	+20.8	33.3	62.3	+26.3 (.43)	+32.7 (.47)	+28.3 (.42)	+28.3 (.44)

Table 10: Per-environment results. "Blocks" is the number of checkpoints competent in the environment, and "All" the gain over all 20 checkpoints (points). The remaining columns are for competent checkpoints: unaided and traced exact success (%), and the gain on each transformation type (points). Mind2Web excludes the six format-only blocks, whose gain is reported in Table 13.

Environment	Blocks	All	Unaided	Traced	D1	D2	D3	D4
AppWorld	7	+6.1	10.0	27.1	+18.0	+11.4	+23.7	+15.4
AgentBench OS	13	+4.9	14.5	21.6	+16.3	+7.5	+2.2	+2.3
InterCode SQL	20	+13.7	50.0	63.6	+22.2	−7.0	+20.9	+18.5
BFCL	19	+11.9	61.8	74.4	+23.8	−3.5	+13.5	+16.4
APIBank	19	+8.5	68.4	77.4	+3.9	+24.1	+0.6	+7.3
Mind2Web	11	+28.7	53.8	68.8	+23.5	+12.9	+8.9	+14.5

M Format Failures in Mind2Web

All 200 native Mind2Web target presentations specify the answer format explicitly, e.g., "Return exactly one JSON object with keys choice, operation, value." The scorer parses the entire completion; a Markdown code fence, an explanation, or a missing or extra key fails the schema, whereas a well-formed object with the wrong element or action is scored and can receive partial credit. The untraced prompt contains no answer example, whereas every source trace ends with a bare JSON answer. Table 13 reports the six checkpoints that fail this format on every untraced Mind2Web target, which we call format-only blocks; all six also score zero on the 50 original Mind2Web sources and are therefore not competent by our criterion. Falcon-H1 1.5B, which solves at least one source, also fails the format frequently (195 of its 200 pairs involve a parse failure in at least one condition) but is not among them. Neither is Qwen3.5 0.8B, which solves no Mind2Web source: 164 of its 200 untraced answers fail the format, but the other 36 are well-formed (2 of them correct); with the trace, 24 fail the format and 30% are correct.

Raw outputs are available for the Falcon-H1 checkpoints only, since the Gemma 3 runs were published as scorer receipts, which bind each output by its hash and record its parse failure, but not its text. Their untraced answers wrap the object in a Markdown code block, sometimes followed by an explanation, whereas their traced answers usually follow the bare format shown in the trace.

Table 11: Gain for competent executors (points) and parameter multiplier per family. The primary multiplier is the environment-adjusted fit, with its 95% interval; the checkpoint-mean fit omits environment intercepts, and the last column excludes pairs with a format failure in either condition.

Family	Blocks	Gain	Multiplier [95% CI]	Checkpoint-mean	Without format failures
Qwen3	33	7.9	1.74× [1.56, 1.95]	2.47×	1.56×
Qwen3.5	26	10.9	1.80× [1.65, 1.98]	1.94×	1.69×
Gemma 3	16	9.1	1.52× [1.41, 1.66]	1.81×	1.26×
Falcon-H1	14	26.3	5.40× [4.40, 6.76]	14.33×	7.35×
Pooled	89	11.9	2.05× [1.90, 2.22]	2.63×	1.85×

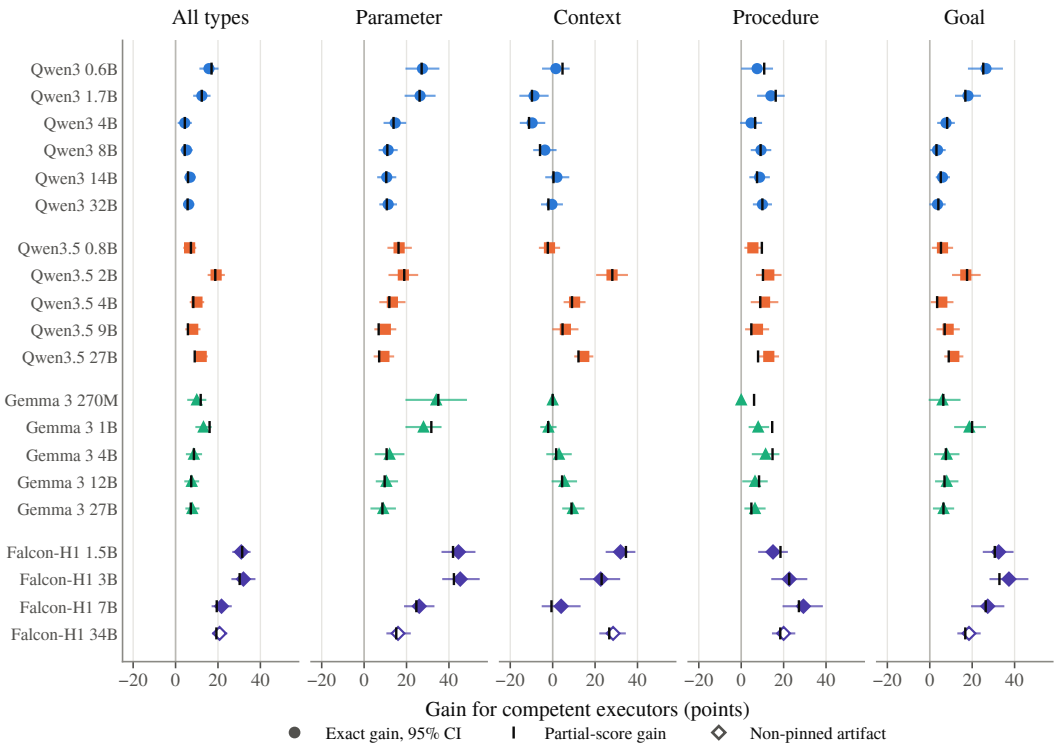

Figure 4: Exact-success gain from trace access for every checkpoint, on the environments in which it is competent, grouped by family and ordered by size, with 95% intervals; vertical ticks mark the partial-score gain. Positive values favor TRACE. The hollow marker is the checkpoint served from a non-pinned artifact.

As a read-only diagnostic, we extracted the contents of a single code block from each available untraced answer and passed them to the unchanged scorer; no stored score was modified. The enclosed answers of Falcon-H1 7B and 34B are mostly correct, so for these checkpoints the trace mainly supplies the format. Falcon-H1 3B also has schema and action errors, and its traced success is lower than that of its enclosed untraced answers. This diagnostic is not a controlled decomposition; separating format from content would require a format-only demonstration, which we did not run.

N RECOVERY AND SCALE WITHIN ENVIRONMENTS

Table 14 reports gains and recovery rates by size tier and transformation type for competent executors. Pooled across environments, recovery on parameter, procedure, and goal changes is 24%, 31%, and 28% by tier (31%, 35%, and 33% with a threshold of five sources). Table 15 reports the same quantity within each family and environment, as a slope on $\log_2$ parameters. Of the 16 estimable

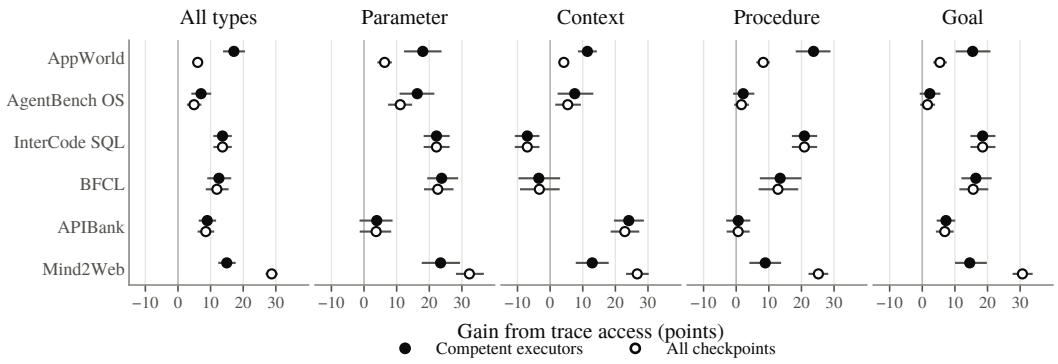

Figure 5: Exact-success gain from trace access in each environment, by transformation type, for competent executors (filled) and over all 20 checkpoints (hollow), with 95% intervals. In InterCode SQL, every checkpoint is competent; in Mind2Web, the all-checkpoint gain includes the six format-only blocks.

Table 12: Exact success on parameter changes without and with the trace (%), and the gain on parameter, procedure, and goal changes (points), over all blocks, split by whether the executor solved the corresponding source unaided. Pair counts are per transformation type.

Tier	Source solved	Pairs	D1: unaided → traced	D1	D3	D4
<2B	yes	211	68.3 → 76.8	+8.5	+1.4	+10.0
	no	1,589	5.7 → 25.1	+19.3	+6.7	+12.8
2–9B	yes	1,032	82.6 → 87.1	+4.6	+5.8	+5.2
	no	1,368	18.4 → 42.0	+23.6	+16.7	+17.7
>9B	yes	1,058	83.5 → 89.1	+5.7	+9.5	+6.4
	no	742	16.0 → 46.6	+30.6	+26.3	+26.8
All	yes	2,301	81.7 → 87.1	+5.4	+7.1	+6.2
	no	3,699	12.5 → 35.7	+23.2	+14.4	+17.4

slopes, 15 are positive, 11 of them with 95% intervals that exclude zero; the one negative slope, for Qwen3.5 on AgentBench OS, also excludes zero. The last column pools each checkpoint's qualifying environments: it controls the family but not the environment mix, and is flat or slightly negative for three of the four families, although, in Qwen3, all five within-environment slopes are positive.

Do procedure changes scale faster? We test whether recovery rises faster with scale on procedure changes than on parameter changes with the difference between their tier slopes (tiers ranked 0, 1, 2), computed within each bootstrap draw. On InterCode SQL, the difference is 0.137 per tier (95% CI 0.023–0.252), and 0.115 (0.004–0.222) with a threshold of five sources. Averaged over the six environments with equal weight, the difference is 0.029, but only 69% of draws define it, chiefly because the gap to the author is too small in BFCL and APIBank, so we do not report an interval. Within family, pooling each checkpoint's environments, the difference per parameter doubling is 0.033 (0.006–0.060) for Qwen3 and 0.033 (0.001–0.067) for Qwen3.5, but these fits do not hold the environment mix fixed, and within-environment differences are heterogeneous (e.g., −0.094 for Qwen3 on AgentBench OS). We therefore claim a steeper rise for procedure changes only where it is identified.

O Original and Revised APIBank Presentation

After observing the APIBank context-change results under the original presentation, we collected two further traced arms on the same 50 targets for all 20 checkpoints. Both use a fixed system wrapper and a single user message with two clearly labeled blocks: a historical block holding the source task and its trace, and a current block holding the complete current task, including its schema and dialogue (Appendix E). In one arm the historical block comes first, in the other last; the two

Table 13: Format-only Mind2Web blocks: targets whose answers fail the required output format, out of 200 per condition, traced exact success (%), and, for Falcon-H1, the share of available untraced answers whose enclosed JSON is exactly correct (diagnostic; available outputs in parentheses). Untraced exact success is zero for all six checkpoints.

Checkpoint	Untraced failures	Traced failures	Traced success	Enclosed correct
Gemma 3 4B	200	167	12.0	–
Gemma 3 12B	200	3	88.0	–
Gemma 3 27B	200	24	82.0	–
Falcon-H1 3B	200	141	19.5	31.6 (196)
Falcon-H1 7B	200	1	87.5	68.0 (200)
Falcon-H1 34B	200	3	90.5	82.3 (147)
All six	1,200	339	63.3	

Table 14: Gain (points) and recovery rate (in parentheses) for competent executors, by size tier and transformation type. The last column pools parameter, procedure, and goal changes before dividing.

Tier	Pairs	D1	D2	D3	D4	D1+D3+D4
<2B	4,200	+28.8 (.37)	+3.7 (.05)	+9.8 (.12)	+19.6 (.25)	+19.4 (.24)
2–9B	7,600	+16.6 (.41)	+5.8 (.11)	+11.8 (.24)	+11.8 (.30)	+13.4 (.31)
>9B	6,000	+11.0 (.34)	+9.1 (.24)	+10.7 (.26)	+8.6 (.24)	+10.1 (.28)
All	17,800	+17.6 (.38)	+6.4 (.12)	+11.0 (.20)	+12.6 (.27)	+13.7 (.28)

differ only in this order. Neither reveals which field changed or its mapping, and the scorer is unchanged, so the obsolete field still fails. Table 16 reports the results. We adopted the history-first arm as the traced condition for these 50 targets in all headline estimates; the original arm and the history-last arm are retained, and Table 8 reports the headline estimates under the original presentation. The substitution is made by condition and target identity, never by outcome, and changes no other target.

We stress three limitations. First, the revision was made after observing the results. Second, the untraced arm was not re-rendered with the new wrapper, so the revised gain compares the new traced presentation with the native untraced one, and the new arms were collected separately from the main run; for Falcon-H1 34B, they also used a different quantization (Appendix F). Third, the comparison between the two labeled arms isolates order, whereas the comparison with the original presentation combines the wrapper, the roles of the messages, and the placement of the current task. Over all blocks, the revision changes the overall gain from 9.4 to 12.3 points, and APIBank's context-change gain from -46.7 to +22.9.

P How Traced Executors Fail

The failure analyses in this paper are automated and structural: they compare recorded calls, terminal reasons, and parser outcomes with the source–target difference, and do not interpret free text. We did not code failure mechanisms by hand, so for the other cases of harm, such as Qwen3 on InterCode SQL, we report outcomes rather than causes.

APIBank obsolete-field audit. For APIBank's context-change targets under the original presentation, we audited every terminal of the first corrected results batch, i.e., both conditions of the 50 targets for the nine checkpoints that batch contained: Qwen3 0.6B, 1.7B, 4B, 14B, and 32B; Qwen3.5 4B, 9B, and 27B; and Falcon-H1 7B. Checkpoints were included because their results were available, not because of their outcomes, and the cohort is not a representative sample of the 20 checkpoints; the shares below describe this cohort only. Unaided, these executors solved 329 of 450 targets (73.1%); with the trace, 56 (12.4%). Of the 394 traced failures, 282 (71.6%) parsed, selected the correct operation, preserved every credited unchanged argument, omitted exactly the renamed field, and supplied exactly the obsolete source field in its place.

Table 15: Within-family slopes of recovery on parameter, procedure, and goal changes against $\log_2$ parameters, per environment, for competent checkpoints (recovery per doubling). *95% interval excludes zero. "–": fewer than two qualifying sizes, or a qualifying checkpoint's gap to the author below 0.05. "Pooled" pools each checkpoint's qualifying environments.

Family	AppWorld	OS	SQL	BFCL	APIBank	Mind2Web	Pooled
Qwen3	.023*	.031*	.058*	.108*	–	.079*	−.013
Qwen3.5	.109*	−.026*	–	.064	–	.175*	.047*
Gemma 3	–	.010	.083*	.073*	.130	–	.012
Falcon-H1	–	–	.138*	.021	.068*	–	−.001

Table 16: Exact success (%) on APIBank's context-change targets for all 20 checkpoints, by size tier, without a trace and under three presentations of the same trace. Headline estimates use the history-first presentation.

Tier	Unaided	Original trace	Labeled, history last	Labeled, history first
<2B	12.7	0.0	7.3	26.7
2–9B	64.8	1.0	67.0	93.5
>9B	75.7	17.7	96.7	99.7
All	52.4	5.7	58.0	75.3

Termination of failed traced attempts. On parameter changes, 2,677 traced attempts fail across all blocks, and a terminal reason is recorded for 1,948 of them: 1,678 (86%) end normally, 242 (12%) at a turn or action limit, 28 for another recorded reason, and none at an output or context limit. Among the 1,018 failures whose calls can be classified, 176 (17%) contain a malformed call.

Output-format audits. The Mind2Web parser audits, including the fenced-answer diagnostic for Falcon-H1 and the audit of Qwen3.5 0.8B, are reported in Appendix M.

Q Extended Related Work

Experience reuse and analogical problem solving. Reusing a previous solution to guide a new one is a longstanding approach to problem solving. Case-based reasoning separates the retrieval, reuse, revision, and retention of prior cases (Aamodt & Plaza, 1994). Derivational analogy further uses recorded problem-solving episodes and their justifications to reconstruct solutions under changed conditions (Carbonell & Veloso, 1988). We study a related question for language-model agents: once a relevant source execution is supplied, can an executor determine which information remains useful? Our traces contain observable actions and environment outputs, rather than explicit symbolic justifications. The executor must therefore infer what to retain, re-ground, or revise. We evaluate this process with fixed source episodes and targets that change arguments, context, procedure, or goal.

In-context demonstrations and their interpretation. Few-shot language modeling (Brown et al., 2020) and chain-of-thought prompting (Wei et al., 2022b) establish that contextual examples can improve performance without updating model parameters. ReAct extends contextual reasoning to agents that interact with external environments (Yao et al., 2022), while analogical prompting generates relevant exemplars before solving a new problem (Yasunaga et al., 2023). Demonstrations can convey more than a transferable solution: they can also specify task format, input distribution, or the space of expected outputs (Min et al., 2022). Their value depends on selection and ordering (Liu et al., 2022; Rubin et al., 2022; Lu et al., 2022). Accordingly, we interpret the paired trace/no-trace difference as the effect of access to a source execution under our delivery policy. The failed-replay requirement establishes that the unchanged source execution is insufficient for the target; it does not independently identify every mechanism by which the additional context helps the executor.

Trajectory demonstrations for language-model agents. Synapse combines state abstraction, trajectory exemplars, and retrieval for computer control (Zheng et al., 2024). TRAD retrieves and

aligns individual demonstration steps (Zhou et al., 2024), while DICE selects examples dynamically with attention to transferable and non-transferable information (Wang et al., 2025a). Particularly close to our setting, Gupta et al. (2025) evaluate trajectory and snippet demonstrations on AppWorld, including transfer from GPT-4o to GPT-4o-mini. Their results already establish that stronger-model trajectories can improve smaller agents. We instead hold one source trace fixed across a broad executor-size sweep and explicitly constructed target transformations. Our purpose is to measure the dependence of trace utility on the executor and the required adaptation, rather than to propose a demonstration-selection algorithm. Patel et al. (2025) also study learning API functionality from demonstrations, especially when documentation is unavailable. Our targets retain their native task and interface information; the intervention adds a related successful execution.

Memory, workflows, and executable skills. Reflexion uses verbal feedback from earlier attempts to guide subsequent trials (Shinn et al., 2023). Rememberer maintains an experience memory (Zhang et al., 2023), ExpeL extracts reusable insights (Zhao et al., 2024a), and CLIN studies continual adaptation across tasks and environments (Majumder et al., 2023). Other methods change the representation of experience before reuse: Agent Workflow Memory induces reusable routines (Wang et al., 2025b), Voyager accumulates executable skills (Wang et al., 2023), and Dynamic Cheatsheet preserves evolving strategies and code snippets (Suzgun et al., 2026). ReasoningBank distills strategies from successful and failed experience (Ouyang et al., 2025), while ACE maintains evolving contextual playbooks (Zhang et al., 2025a). These systems demonstrate complementary ways of turning experience into useful guidance. We fix the supplied episode and its delivery instead of jointly optimizing memory construction, retrieval, and representation. Our estimates therefore concern trace-conditioned execution, rather than the full lifecycle of a deployed memory system. This distinction also separates reading an episode from invoking code that already implements part of its procedure.

Benchmarks of procedural transfer and continual learning. Recent benchmarks increasingly evaluate experience reuse directly. Evo-Memory compares self-evolving memory over task streams (Wei et al., 2025). AgentCL constructs explicit cross-task relationships and separates plasticity, stability, and held-out generalization (Shu et al., 2026). AFTER evaluates procedural skills across tasks, roles, and model backbones (Belikova et al., 2026). EvoAgentBench constructs trace-grounded procedural relationships, ensures prior support for test tasks, and evaluates transfer relative to unevolved agents (Gao et al., 2026). Thus controlled task relationships and cross-model procedural transfer are established evaluation concerns. Our setting contributes a more specific unit of analysis: the paired effect of one verified source trace on an executor facing one of four specified changes. The source episode is shared across executors and targets, allowing trace utility to be related to executor scale and to observed unaided source performance. This intervention complements stream-level measures of memory accumulation and retention; it does not itself evaluate either process.

Adaptation after a change and harmful reuse. PACE-Bench constructs physics-design pairs in which a successful source solution fails after an environment change, and evaluates iterative adaptation with simulator feedback (Zhan et al., 2026). It provides a close precedent for requiring more than replay, while differing from our trace-access intervention and software-task setting. More generally, Xiong et al. (2026) identify experience-following behavior, error propagation, and misaligned replay. Hu et al. (2026) compare raw trajectories with abstract insights on controlled task transitions and show that procedural detail can become harmful under transfer. We therefore do not treat negative transfer as a new phenomenon. Our verified source outcomes and specified transformations help distinguish a source that was unsuccessful from one whose successful behavior no longer applies. Prompt order, irrelevant context, and the position of information can also affect context use (Lu et al., 2022; Shi et al., 2023; Liu et al., 2024). Our interpretation of harmful transfer remains conditional on delivery, particularly when a historical interface conflicts with the current one.

Transfer between models and contextual teaching. Knowledge distillation transfers supervision into student parameters (Hinton et al., 2015); reasoning distillation uses teacher-generated reasoning or rationales to train smaller models (Magister et al., 2023; Hsieh et al., 2023). Our executors are frozen. In-context teaching is a closer precedent: Zhao et al. (2024b) provide teacher-generated rationale demonstrations for unseen reasoning cases. Trace-probing studies also inject partial reasoning into another model (Ballon et al., 2026). Our setting instead supplies observable executions of a

different, related task, without the author's hidden reasoning or a target solution. Beyond language agents, trajectory-conditioned policy generalization (Xu et al., 2022) and Algorithm Distillation (Laskin et al., 2023) establish demonstrations and learning histories as inference-time adaptation resources. Our emphasis is on measuring this form of conditioning in existing instruction-tuned language models across diverse tool interfaces, rather than training a new adaptation mechanism.

Executor scale and the ability to use context. The ability to benefit from contextual information need not increase uniformly across tasks or models. Wei et al. (2023b) show scale-dependent differences in learning contextual mappings and overriding semantic priors, and Shi et al. (2024) study related behavior theoretically. Symbol tuning demonstrates that training can improve the use of contextual mappings (Wei et al., 2023a). These findings motivate measuring the borrower as well as the supplied information. We examine how observable source executions help checkpoints of different sizes, using target success rather than label-mapping performance. Language-model scaling laws provide broader context (Kaplan et al., 2020; Hoffmann et al., 2022), but do not imply that agentic success is determined by parameter count alone. Our scale comparisons are empirical and within-family; they do not identify the causal effect of adding parameters while holding training and architecture fixed.

Measuring gains and resource equivalence. Matched improvements and normalized gains have precedents in agent evaluation. Continual Learning Bench compares stateful and stateless execution and normalizes improvements by available headroom (Asawa et al., 2026). Our recovery measure uses the author's target performance as an empirical reference rather than a maximum possible reward. Parameter-equivalence follows the resource-matching perspective of Le Scao & Rush (2021), who express prompting improvements in equivalent training examples. We instead express the benefit of a source trace in model-size units, through a log-linear fit of unaided and traced success within each family. Retrieval-augmented language modeling already establishes external information as a complement to model parameters (Lewis et al., 2020; Borgeaud et al., 2022). We measure a particular external resource—one verified related-task episode—and report a conditional performance equivalence, not an equivalence in inference cost or general capability.

Deployment and the cost of assistance. Small-model deployment motivates examining how much capability can be supplied externally (Belcak et al., 2025). FrugalGPT and RouteLLM select among models to manage serving cost (Chen et al., 2023; Ong et al., 2024), while Minions studies active collaboration between local and cloud models (Narayan et al., 2025). Agentic Plan Caching reuses structured plans and employs lightweight models for adaptation (Zhang et al., 2025b). These systems motivate, but do not directly measure, the intervention studied here: a stored trace offers information without live assistance from its author on the target. Reusing it can avoid weight updates or a new author invocation, but still incurs context-processing costs. Executable alternatives, including recent work on reusable agent harnesses (Li et al., 2026), further emphasize that experience can improve a deployed system through mechanisms other than the executor reading a raw episode. We study that episode-conditioned executor as one component of the broader design space.

Relationship to the source benchmarks. AppWorld, AgentBench, InterCode, BFCL, API-Bank, and Mind2Web provide the task interfaces from which our tasks are derived (Trivedi et al., 2024; Liu et al., 2023; Yang et al., 2023; Patil et al., 2025; Li et al., 2023; Deng et al., 2023). Transformation-based behavioral evaluation has a broader precedent in CheckList (Ribeiro et al., 2020). Our tasks add reviewed source–target relationships, non-replayability checks, and matched trace-access conditions. Because tasks and some checkers are adapted, the resulting measurements are scores under our versioned checkers, rather than reproductions of the upstream leaderboards. The distinction between bounded static responses and stateful execution also remains relevant when interpreting transfer across the six environments.