TRAJECTORIES AS PROGRAMS: VERIFIED REPLAY AND STATE RECOVERY FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents are typically given multiple attempts at the same task. A failed attempt has often already completed part of the task. To reuse past attempts, existing methods turn them into text and ask the model to regenerate the actions from the initial state. The regenerated actions may be different from the original ones, so text cannot bring the task exactly back to a state the previous attempt reached, and the completed part cannot be kept. To address this problem, we propose Trajectories as Programs (TaP), a training-free framework that stores each attempt as an executable program, so a new attempt can start exactly where the completed part ends instead of regenerating everything from text. To decide where a new attempt should start, we introduce Verified State Localization: it replays the program of a failed attempt and keeps the states that have verifiably completed part of the task as recovery points. But recovery points are not equally useful, and attempts are limited. To spend the limited attempts on the recovery points most likely to solve the task, we introduce Probe-Guided Attempt Allocation: cheap probes first judge which recovery points can leave the old failed path and make new progress, and full attempts go to the most promising recovery points. Once a task is solved, TaP replays its verified program directly instead of calling the model, so the task stays solved and every new attempt goes to the tasks that are still unsolved. Across AppWorld and BFCL-V3 and across model sizes, TaP sets a new state of the art on both metrics: it solves more tasks per attempt on average, because a solved task never has to be solved again, and solves more tasks at least once over all attempts, because new attempts start from verified recovery points.

1 INTRODUCTION

Large language model (LLM) agents can complete complex tasks by taking actions in an environment and observing the results (Yao et al., 2023b; Wang et al., 2024a). A single attempt at a task rarely succeeds, so in practice an agent is usually allowed to try the same task multiple times (Shinn et al., 2023). Most of these attempts still fail, but a failed attempt is not worthless: it may have correctly completed most of the task, only to fail at the last few steps.

To reuse a failed attempt, existing methods convert the attempt into text, such as a summary of what was tried or a reflection on why the attempt failed (Shinn et al., 2023; Zhao et al., 2024; Zheng et al., 2024; Wang et al., 2024c). Then the model reads the text and regenerates the actions from the initial state. But the model does not faithfully follow the experience described in the text: at the same step, it may take a different action (Figure 1, bottom left). Text therefore cannot bring the task exactly back to a state the previous attempt reached, so the completed part of a failed attempt is lost, and text can even make a solved task fail again. This is the bottleneck of the whole line of work on reusing experience across attempts: not how much experience is stored, but the form in which it is stored. Experience is described rather than executed. Even WikiSkill (Tang et al., 2026), which compiles past attempts into a persistent wiki and reusable skills, stores the skills as text that the model reads before regenerating the actions, so it does not step outside this framework.

To address this limitation, we propose Trajectories as Programs (TaP), a training-free framework that stores each attempt as an executable program that records the exact actions of the attempt. When the environment can be reset and is deterministic, executing the program reproduces exactly the same

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

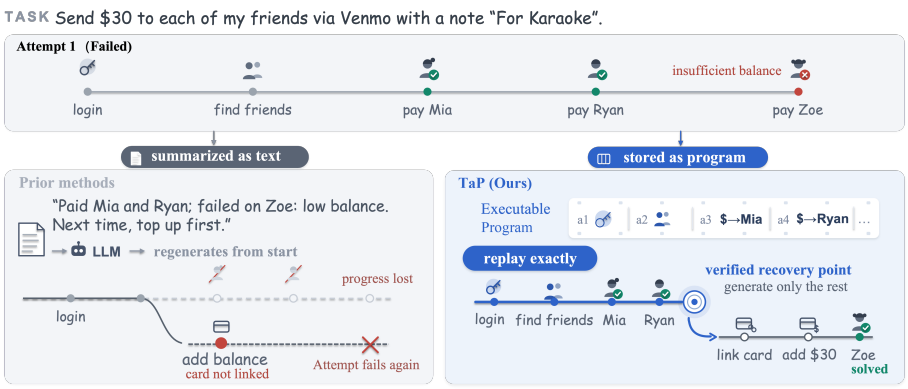


Figure 1: Executable program replay rather than textual regeneration. The task is to send $30 to each Venmo friend. Attempt 1 pays Mia and Ryan but fails on Zoe because the balance is insufficient. Prior methods summarize the attempt as text and regenerate from the initial state; the regenerated actions differ, the completed payments are not reproduced, and the attempt fails again. TaP replays the stored program to restore exactly the verified state after Ryan is paid and generates only the remaining actions, which solve the task.

states as the previous attempt, so TaP can precisely restore any state the previous attempt reached and let a new attempt continue from the middle of the task instead of starting from the beginning. Specifically, to locate the recovery points that are truly useful to the task, we introduce Verified State Localization: it executes the program of a failed attempt, checks the task after every action, and keeps the states that are verified to have completed a new part of the task as recovery points.

However, recovery points are not equally useful, and the attempts are limited for a certain task. A late recovery point keeps more completed work, but the model often walks back down the old failed path and fails in the same way; an early recovery point leaves the old failed path more easily, but keeps less completed work. To judge which recovery point deserves an attempt, we introduce Probe-Guided Attempt Allocation: before spending any full attempt, TaP launches cheap probes from the recovery points, and a probe generates only a few actions and tests whether the model can leave the old failed path and make new progress from that recovery point. Full attempts then go to the most promising recovery points, so the limited attempts are spent on the recovery points most likely to solve the task.

Once a task is solved, the model does not need to generate the actions again. TaP removes the unnecessary actions from the program, executes the shortened program to verify that it still solves the task, and then replays the verified program directly in later attempts without calling the model. The solved task therefore stays solved and costs no further attempts. Across AppWorld (Trivedi et al., 2024) and BFCL-V3 (Patil et al., 2025) and across model sizes, TaP outperforms strong memory baselines and sets a new state of the art on both metrics: it solves more tasks per attempt on average, and solves more tasks at least once over all attempts.

2 Related Work

Reusing experience across attempts. Most methods reuse experience as text: a reflection on why the last attempt failed (Shinn et al., 2023), insights distilled from past tasks (Zhao et al., 2024), past attempts retrieved as exemplars (Zheng et al., 2024), or workflows induced from past attempts (Wang et al., 2024c). In all of them the model reads the text and regenerates the actions, so a state reached by a previous attempt cannot be restored. Voyager (Wang et al., 2023a) and TroVE (Wang et al., 2024b) store experience as code, but as building blocks: the model still writes new code in the next attempt, so reuse again goes through regeneration. TaP instead replays the stored program of an attempt directly, so the exact states of the previous attempt are restored rather than regenerated.

Spending multiple attempts. Tree-search agents branch from intermediate states and backtrack within one attempt (Yao et al., 2023a; Zhou et al., 2024), but the tree lives only inside that attempt. Go-Explore (Ecoffet et al., 2021) and its foundation-model version (Lu et al., 2025) archive states and return to them for exploration, restoring a state from a simulator snapshot and choosing where

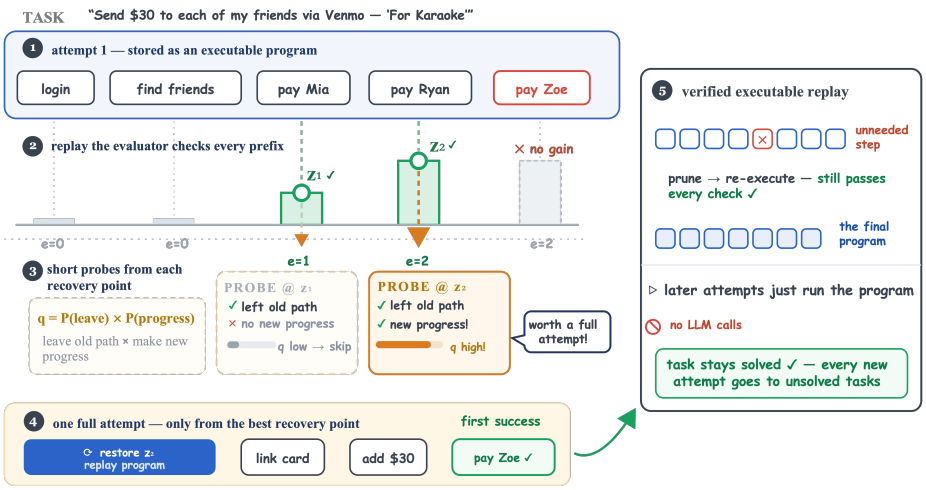


Figure 2: Overview of TaP. Given a failed attempt, we (1) store it as an executable program; (2) replay the program, run the evaluator on every prefix, and keep the earliest depths that reach a new progress high as recovery points z_1 and z_2; (3) score each recovery point by short probes, $q = \rho g$ being the probability of leaving the old path times the probability of improving given that it left (P(leave) and P(progress) in the figure, Eq. 4), which selects z_2; (4) spend a full attempt from z_2, replaying $a_1, \ldots, a_4$ and generating only the remaining actions, which solve the task; and (5) prune the successful sequence C into a shorter program P that still passes every check, which later attempts execute without calling the model.

to return by novelty; TaP restores a state by executing the recorded program, keeps only states whose progress the task checker has verified, and spends a fixed attempt budget by probing. Tree search over web pages (Koh et al., 2024) backtracks within one episode, as LATS does, and Skill-Weaver (Zheng et al., 2025) distils practice into callable skills, the regeneration route of Voyager. Sampling many independent attempts scales test-time compute (Wang et al., 2023b; Brown et al., 2024; Snell et al., 2024), but every sample starts from the initial state, while TaP starts from the verified recovery points of failed attempts.

3 Method

An agent π_θ receives a task and interacts with an environment. At each step t, the agent outputs an action a_t, and the environment returns an observation o_t. We write $\tau = (a_1, o_1, \ldots, a_L, o_L)$ for the trajectory of an attempt that ends after L steps. TaP turns a trajectory τ into an executable program. A new attempt can then start from a state in the middle of τ that has already completed part of the task, instead of starting from the beginning. When τ solves the task, later attempts execute the program, so the task stays solved. Figure 2 gives an overview.

3.1 Storing Trajectories as Programs

To reuse past experience without losing any details, TaP stores each trajectory τ as an executable program, together with the model context that produced its actions. An action is a block of executable code, so the program of τ is the code blocks the attempt ran, in order. Storing the program rather than a description of it is what makes the record faithful: replaying the code blocks reproduces the observations the attempt saw, so any state along τ can be reached again exactly, and the stored context lets a new attempt continuing from the middle of τ see the same interaction history the original attempt saw at that point. We assume that the environment is resettable and deterministic. A reset returns the environment to the initial state s_0, and executing the same action prefix after a reset reaches the same intermediate state.

Each task has an evaluator with a set A of checks. The evaluator runs on an environment state s and returns the set of passed checks $E(s) \subseteq A$. For a trajectory τ, we write $E(\tau)$ for the checks passed at its final state. We define the success label as $y_\tau = \mathbf{1}[E(\tau) = A]$ and the partial score as $p_\tau = |E(\tau)|/|A|$. Here, y_τ indicates whether the task is fully solved, while p_τ measures the

progress made by the trajectory. Evaluator results are used only to select and verify programs, and evaluator text is never placed in the model's context.

3.2 Verified State Localization

A failed trajectory may still contain useful progress, while some later states may no longer be reliable because of an earlier mistake. Restarting from the initial state discards all useful work, but restarting too late may reproduce the same failure. We therefore measure progress along the trajectory and keep a small set of recovery points that preserve verified work while avoiding the failed later steps.

Tracking Progress Along Failed Trajectories. For each unsolved task, TaP keeps a small frontier of failed trajectories. Trajectories with higher final partial scores are preferred. If two trajectories have the same partial score, we keep the longer one because it offers more recovery points. For each new frontier trajectory, TaP resets the environment, replays its actions, and evaluates every prefix. For the prefix $\tau_{1:d}$ ending at depth d, we record

$$e_d = \left| E(\tau_{1:d}) \setminus E(s_0) \right|, \tag{1}$$

where $E(s_0)$ is the set of checks already passed in the initial state s_0. Thus, e_d counts the new checks earned by the first d actions, and a larger e_d means that more verified work has been completed. The sequence $(e_1, \ldots, e_L)$ forms the measured progress profile.

Selecting Recovery Points from Measured Progress. Many nearby prefixes pass the same checks. Keeping all of them would waste exploration budget, so we retain only the first depth that reaches each new progress high. Let $\mathcal{D}(\tau)$ be the selected recovery depths:

$$\mathcal{D}(\tau) = \left\{ d < L : \; e_d > \max_{0 \le r < d} e_r \right\}, \qquad e_0 = 0. \tag{2}$$

Here, r indexes earlier depths. A depth d is selected only when e_d exceeds every earlier value. Choosing the earliest depth at a new high preserves the verified work and leaves the largest remaining action budget. Action prefixes that are the same are merged because deterministic replay reaches the same state. As a result, $\mathcal{D}(\tau)$ retains the earliest usable depths that reach new verified progress.

Recovering a Saved State. Let $z = (\tau, d)$ denote a recovery point. Its action prefix is $C_z = (a_1, \ldots, a_d)$, and h_z is the matching message history. We reset the environment, replay C_z, restore h_z as the model context, and generate only the suffix:

$$s_z = \mathrm{exec}(C_z), \qquad \tau' = C_z \oplus \mathrm{Gen}(\pi_\theta; s_z, h_z, T). \tag{3}$$

Here, s_z is the recovered environment state, T is the sampling temperature, Gen generates the new action sequence, and $\oplus$ joins the replayed prefix with the new suffix. Recovering both s_z and h_z lets the model continue from the same environment state and interaction context.

3.3 Probe-Guided Attempt Allocation

A useful recovery point must allow the model to leave the old failed path, and the new path must lead to more progress. TaP estimates these two properties with short probes and spends full attempts on the most promising choices.

Measuring the Value of a Recovery Point. For each recovery choice (d, T), we estimate whether it can leave the previous trajectory and whether the new trajectory makes better progress.

For each task, TaP probes at most four recovery points, taking those with the highest verified progress first, and runs m probes per choice, each generating at most k actions from the recovered state. Let $D = 1$ if a probe produces an observation sequence different from the recorded sequence at the corresponding depths; probes with no observation are discarded. We compare observations because different actions may have the same effect. For a deviating probe, let $I = 1$ if its final progress e exceeds the recorded progress at the same depth. For a full attempt, we instead compare its final e with the final e of the recorded trajectory; a successful full attempt therefore has $I = 1$. We define

$$q(d, T) = \rho(d, T) g(d), \qquad \rho(d, T) = \Pr(D = 1 \mid d, T), \qquad g(d) = \Pr(I = 1 \mid D = 1, d). \tag{4}$$

Here, $\rho(d, T)$ is the probability of leaving the old path, and $g(d)$ is the probability of improving after leaving it; their product $q(d, T)$ scores the recovery choice. The two factors move in opposite directions along the trajectory: a deeper point holds more verified work but conditions the model more strongly on the failed path, so ρ falls with depth while g rises, and their product locates the depth at which the two balance. Since ρ depends on the sampling temperature and g on the state, we estimate $\rho(d, T)$ separately for each temperature and pool $g(d)$ across temperatures at the same recovery depth, using frequency counts with a smoothing prior.

Updating Recovery Points. Evidence from one recovery point also changes the search priors of other points on the same trajectory. When an attempt from depth d leaves the recorded path and improves progress, TaP raises the prior value of all shallower recovery points; it does not add artificial observations to their statistics. If a recovery point produces at least four deviating but non-improving attempts across at least two temperatures, and no positive evidence has established it as usable, TaP removes that point and all deeper candidates. These updates affect only the search priors and candidate set, not the recorded success counts.

Allocating Full Attempts. The budget is a single global pool whose size is a fixed number of full attempts per unsolved task, and it is spent one full attempt at a time by marginal value, without a per-task quota. For each choice $u = (i, d, T)$, where i denotes the task, we maintain a Beta distribution over its solve probability:

$$\pi_u \sim \mathrm{Beta}\left(c_0 q(d, T),\ c_0[1 - q(d, T)]\right),$$

where $q(d, T)$ is the prior mean and c_0 controls the prior strength. The distribution is updated only using full-attempt outcomes. At the start of an allocation we draw one sample $\widehat{\pi}_u$ per choice, and then fill the pool by repeatedly selecting

$$u^* = \underset{u=(i,d,T)}{\arg\max}\ \widehat{\pi}_u\, \rho(d, T)^{n_u} \phi_i, \qquad \phi_i \leftarrow \phi_i \left(1 - \widehat{\pi}_{u^*}\right). \tag{5}$$

Here, n_u is the number of full attempts the current allocation has already given to u, and ϕ_i, initialized to 1, estimates the probability that task i remains unsolved and is updated whenever a choice for that task is selected. Thus, $\widehat{\pi}_u$ favors choices likely to solve the task, $\rho(d, T)^{n_u}$ penalizes repeated scheduling, especially choices that frequently return to the old path and ϕ_i directs more exploration toward tasks likely to remain unsolved. Once a task is solved, all its choices are removed. Full-attempt outcomes update the deviation and progress statistics, while any probe or full attempt that solves the task is passed to the replay step in Section 3.4. Probes, full attempts, and the offline prefix evaluations are extra interactions beyond the attempt budget of the baselines; we count all of them and report the totals in Section 4.

3.4 Verified Executable Replay for Solved Tasks

For a solved task, generating a new trajectory may fail because the sampled actions can differ from those that previously succeeded. To avoid this regression, TaP converts the first successful trajectory, either from the initial state or from a recovery point, into a verified executable replay program. Under resettable deterministic execution, replaying this program reproduces the successful behavior without further model calls.

Let C be the successful action sequence and $E(\tau)$ its passed checks. We simplify C by first removing large parts of the action sequence and then individual actions. A deletion Δ is kept only if re-execution in a reset environment still verifies

$$E\big(\mathrm{exec}(C \setminus \Delta)\big) \supseteq E(\tau). \tag{6}$$

Actions that explicitly complete the task are not removed. Pruning matters for replay itself: a raw successful trajectory carries the exploratory actions that preceded the solution, which lengthen every later replay and can exceed the interaction budget, whereas the pruned program keeps only the actions the checks depend on. The final program is accepted only if re-execution still succeeds; otherwise, we use the original successful sequence. The accepted program is stored as P_i and executed directly, without calling the model, in every later evaluation, so solved tasks remain solved and the budget goes to unsolved tasks.

Table 1: Main results. Avg@4 and Pass@4 (%) on AppWorld and BFCL-V3 for four backbones; mean and sample standard deviation (small) of three independent runs. Best in bold, second best underlined.

	Qwen3-8B		Qwen3-14B		Qwen3-32B		Mistral-Small-3.2-24B	
Method	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4
AppWorld (168 tasks)								
No Memory	14.97±0.24	32.85±2.11	22.57±0.19	41.07±0.84	27.23±0.92	50.59±1.68	17.41±0.45	35.32±0.69
A-Mem	12.95±0.37	29.76±2.80	18.95±0.31	37.70±0.57	28.13±0.75	51.19±0.97	14.58±0.74	31.94±1.72
LangMem	11.46±0.53	26.79±0.84	21.88±1.37	41.67±1.68	24.55±0.57	47.02±1.56	14.68±0.48	31.15±0.91
ReMe (fixed)	17.06±0.25	36.31±1.29	25.35±0.91	46.82±0.74	31.50±0.67	58.13±1.40	19.74±0.37	39.48±0.91
Reflexion	17.76±0.60	35.52±0.91	26.79±0.30	45.44±1.37	31.94±0.34	54.96±0.91	20.68±0.39	38.29±0.91
Memp	18.80±0.62	36.71±1.72	28.22±0.82	46.43±1.19	33.43±0.76	55.75±1.50	21.88±0.79	39.29±0.60
WikiSkill	19.64±0.79	37.10±0.91	29.32±0.30	48.21±1.19	35.47±0.31	57.94±1.24	22.77±0.39	40.67±1.37
ReMe (dynamic)	24.70±1.04	42.06±0.74	34.32±0.81	52.98±1.29	42.02±0.51	63.49±0.28	27.58±0.37	45.44±1.24
TaP (ours)	**34.47**±0.62	**48.02**±1.50	**51.69**±0.70	**65.48**±1.19	**54.66**±0.56	**70.04**±0.91	**40.43**±0.67	**53.97**±1.50
BFCL-V3 (150 tasks)								
No Memory	40.33±0.94	59.55±0.83	48.66±1.51	68.22±0.63	54.55±0.63	72.44±0.83	42.00±0.87	60.89±1.68
A-Mem	41.22±0.61	62.00±2.37	47.44±0.44	69.77±0.63	54.50±1.09	72.66±0.54	42.39±0.75	63.33±1.33
LangMem	44.11±0.28	65.55±1.13	49.17±0.33	71.33±1.33	52.27±1.13	72.22±1.91	45.00±0.60	66.22±1.02
ReMe (fixed)	44.50±0.85	65.77±0.63	51.89±0.34	72.44±0.63	56.05±1.26	74.89±0.63	45.83±0.58	66.44±1.02
Reflexion	42.33±0.87	63.11±0.77	51.50±0.67	71.11±1.02	55.11±0.54	73.78±1.02	44.28±0.63	64.22±1.39
Memp	43.67±0.60	65.56±0.77	53.06±0.38	72.89±1.02	55.89±0.69	75.78±1.02	45.78±0.25	66.67±0.67
WikiSkill	44.22±0.42	66.22±1.02	54.28±0.63	74.22±1.68	55.83±0.44	76.22±1.02	46.39±0.51	67.33±1.15
ReMe (dynamic)	45.17±0.36	68.00±0.55	55.00±0.72	74.44±0.83	56.17±0.24	76.44±1.13	47.11±0.51	69.33±1.15
TaP (ours)	**60.11**±0.54	**70.44**±1.02	**69.72**±0.48	**81.11**±1.02	**71.44**±0.42	**80.00**±0.67	**62.39**±0.51	**71.78**±1.02

4 EXPERIMENTS

4.1 SETUP

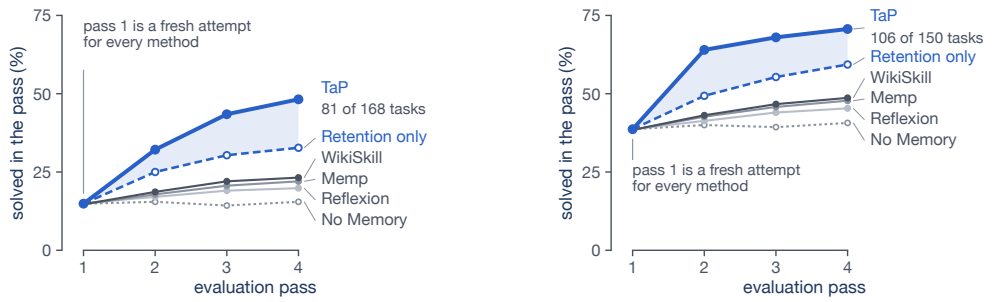
Benchmarks and protocol. We evaluate on AppWorld (Trivedi et al., 2024) (`test_normal`, 168 tasks) and BFCL-V3 (Patil et al., 2025) (`multi_turn_base`, 150 tasks) under the protocol of ReMe (Cao et al., 2025): every method makes four *evaluation passes* over the task set, a pass being one attempt at every task with at most 30 interactions (Appendix A). Avg@4 is the success rate averaged over the four passes and Pass@4 the fraction of tasks solved in at least one pass. Memory baselines build their pools on the training split before the first pass, as in ReMe; TaP starts every run with an empty bank, so its first pass is an ordinary fresh attempt.

Backbones and baselines. We use Qwen3-8B, Qwen3-14B, Qwen3-32B (Yang et al., 2025) and Mistral-Small-3.2-24B (Mistral AI, 2025), and GPT-4o (OpenAI, 2024), DeepSeek-V3 (DeepSeek-AI, 2024) and Gemini 2.5 Pro (Comanici et al., 2025) through their APIs. We compare with No Memory, the text memories A-Mem (Xu et al., 2025), LangMem (LangChain, 2025), Reflexion (Shinn et al., 2023), Memp (Fang et al., 2025) and WikiSkill (Tang et al., 2026), and ReMe (fixed and dynamic). On Qwen3 the No Memory, A-Mem, LangMem and ReMe rows are the numbers ReMe reports under this protocol; the other rows are run by us under it.

TaP configuration and accounting. The frontier holds four failed trajectories per unsolved task, at most four recovery points per task are probed with $m = 2$ probes of $k = 5$ actions each, $c_0 = 1$, and the global pool holds nine full attempts per unsolved task per round; probes and full attempts from recovery points draw T from $\{0.7, 0.9, 1.1\}$ on AppWorld and $\{0.9, 1.1, 1.3\}$ on BFCL-V3 (Appendix A). Table 1 and the closed-model results are means and sample standard deviations of three runs; per-pass curves, ablations and sweeps use one of them, the *reference run*, with integer task counts. Every probe, full attempt and prefix evaluation is counted: 130.43–376.29 model calls per task over the protocol (Appendix B). TaP reads the checker on intermediate states, the baselines at the end of an attempt, and no checker output enters a prompt.

4.2 MAIN RESULTS

TaP is first on both metrics in all sixteen cells. In Table 1, TaP improves Avg@4 over the strongest baseline, ReMe (dynamic), by 9.77–17.37 points on AppWorld and 14.72–15.28 on BFCL-V3, and Pass@4 by 5.96–12.50 and 2.44–6.67 points. Pass@4 counts a task once, so it rises only



(a) AppWorld (b) BFCL-V3

Figure 3: A solved task is never lost, so the curve rises in every pass. Success rate per evaluation pass with Qwen3-8B; pass 1 is a fresh attempt for every method. *Retention only* replays solved tasks without the recovery search; the shaded band is what the search adds (all settings: Appendix Figure A1).

when a task no fresh attempt solves is solved from a recovery point, whereas Avg@4 also rewards keeping a task solved in every later pass; Section 4.5 separates the two.

The separation opens after pass 1 and never closes. In Figure 3, from pass 2 on the curve of TaP rises at every step, to 48.21 on AppWorld and 70.67 on BFCL-V3, while the text baselines flatten after pass 2. Retention only flattens too: the fresh re-solve rate of the remaining tasks decays from 11.89% to 3.42% over the three retry rounds on AppWorld (Appendix F). A replayed task costs no model call and 7.3 s against 18.1 calls and 79.1 s for regenerating it, and by pass 4 the bank supplies 96.3–99.2% of the score (Appendix Tables A3 and A17).

Attempts from recovery points buy more than the same budget from the initial state. Given the model calls of TaP as additional independent passes, No Memory (14 to 24 passes) and ReMe (dynamic) (11 to 20) end 17.15–29.10 and 6.35–13.00 Avg@K points below the Avg@4 of TaP, and 2.22–18.65 and 1.77–9.53 Pass@K points below its Pass@4 (Appendix Table A4): the first solves of attempts from the initial state decay geometrically from pass to pass (No Memory gains 2.58–6.45 Pass points from pass 4 to the matched K, ReMe 0.67–3.57), while an attempt from a recovery point starts with part of the task already verified. Search inside an attempt (LATS-style (Zhou et al., 2024), ToT-style (Yao et al., 2023a)) at equal interactions per pass raises the per-pass rate, stays flat across passes and ends 2.83–5.50 Avg@4 and 3.34–8.92 Pass@4 below TaP (Table A9).

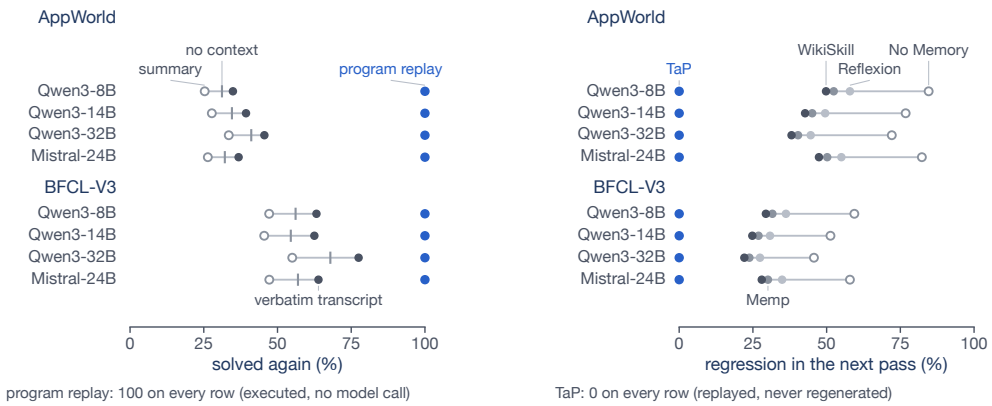
The method runs unchanged on closed models. TaP consumes only sampled actions and checker verdicts, so it applies unchanged to GPT-4o, DeepSeek-V3 and Gemini 2.5 Pro, raising Avg@4 by 14.44–18.31 points over No Memory and Pass@4 by 9.33–16.47 (Appendix G); the 121 programs banked by GPT-4o replay on Qwen3-8B without loss, 72.02 Pass@4 on AppWorld before any model call (Appendix Table A20).

4.3 Text Describes an Attempt; a Program Restores It

The same experience re-solves the task 25–78% of the time as text and 100% as a program. Figure 4a isolates the form of experience from its content. The three regenerating forms re-solve 25.31–77.5% of the tasks, the summary is below the empty context on every row, and the program re-solves 100%: a description of an attempt does not restore it, and the regenerated actions diverge from the recorded ones early (Appendix Figure A6).

Text memories lose a quarter to a half of what they solve; replay loses nothing. Among tasks solved in one pass (Figure 4b), Reflexion, Memp and WikiSkill fail 22.1–57.9% in the next pass, against 45.7–84.6% for No Memory; TaP regresses on none, and replacing its replay with regeneration from a transcript or a summary brings the regression to 36.79–74.69% (Appendix Table A26).

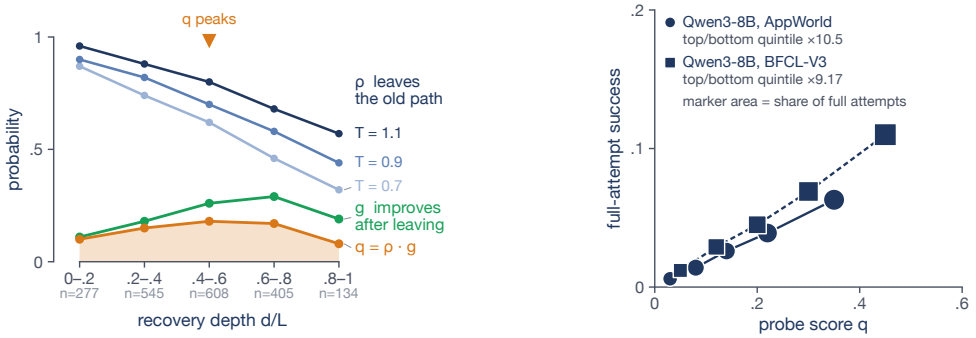
What a program carries is the state, not the feedback. Given the per-check feedback that TaP reads, Reflexion and Memp gain 1.6–1.9 Avg@4 and 1.1–2.0 Pass@4 points (Appendix Table A25): the regenerated actions still differ from the ones that earned the checks, so the completed part is lost.



(a) The same successful experience in four forms

(b) Solved tasks lost in the next pass

Figure 4: The same experience as text and as a program. (a) Every task solved by TaP is attempted again with its own successful attempt given as an LLM-written summary, as the verbatim transcript, or as nothing in the context (four samples per task, regenerated from the initial state), or by executing the stored program. (b) Share of (task, pass) pairs solved in pass k and failed in pass $k{+}1$, pooled over the three transitions, each method on its own solved set.



(a) Recovery-depth trade-off

(b) Probe scores against realised success

Figure 5: Recovery has an interior optimum, and probes find it. (a) All probes of TaP (Qwen3-8B, AppWorld), binned by the relative depth d/L of their recovery point, n per bin below the axis: ρ, the probability that a probe leaves the recorded observation sequence; g, the probability that a leaving probe improves on the recorded progress; and their product $q = \rho g$ (ρ at $T = 0.9$), whose peak is marked (▼). (b) Recovery choices that received full attempts, in quintiles of q, against their realised success rate (Qwen3-8B); marker area is the share of full attempts spent on the quintile.

4.4 Where to Recover, and Which Recovery Point to Spend On

Most failures contain verified work, and it sits in the middle of the trajectory. Among the pass-1 failures of the reference run, 58.04–76.47% have earned at least one check beyond the initial state, and the last progress event of the median progressed failure sits between $0.42L$ and $0.55L$ (Appendix D): a restart in the middle exists for most failures; the question is which depth.

Late points keep the work but repeat the failure; early points leave it but keep less. Figure 5a measures both quantities of Eq. 4 on every probe. The probability ρ of leaving the recorded path falls monotonically with depth; the probability g of improving after leaving rises with depth until the last bin, where the state usually already contains the damaging action; and their product q peaks in an interior bin on all eight settings (Appendix Figure A4). Restarting from the last step before the failure is therefore the worst rule we compare (28.72 Avg@4 on AppWorld, Appendix Table A22).

Cheap probes predict expensive attempts. Over the recovery choices that received full attempts, realised success rises monotonically across quintiles of q on every setting (Figure 5b and Appendix Figure A6; Spearman 0.57–0.66), and the top quintile succeeds 7.67–13.50 times more often than the

Table 2: Component ablation (Qwen3-8B). Each row changes one component of the configuration of Table 1; the small fraction is the number of tasks solved at least once in the reference run. The full model and the four search rows are three-run means ± one standard deviation; the other rows are from the reference run. Per-pass counts and further rules: Appendix H.

	AppWorld		BFCL-V3	
Variant	Avg@4	Pass@4	Avg@4	Pass@4
TaP (full)	**34.47**±0.62	**48.02**±1.50 81/168	**60.11**±0.54	**70.44**±1.02 106/150
w/o Verified State Localization (geometric depths)	32.04±0.70	45.04±1.50 76/168	56.56±0.67	65.78±1.02 99/150
w/o probes (prior from partial score only)	33.33±0.68	46.43±1.19 78/168	58.83±0.60	69.11±1.02 104/150
w/o marginal allocation (UCB over choices)	33.43±0.60	46.23±1.50 78/168	59.00±0.58	69.11±1.02 104/150
w/o context restore (environment state only)	33.33±0.65	47.02±1.19 79/168	59.39±0.51	69.78±1.02 105/150
w/o pruning (raw successful program replayed)	34.52	48.21 81/168	60.33	70.67 106/150
w/o verified replay (solved tasks regenerated)	19.20	48.21 81/168	41.67	70.67 106/150
skill-library reuse (programs as callable functions)	22.02	48.21 81/168	51.50	70.67 106/150
text-only experience (attempt stored as a summary)	16.22	38.10 64/168	41.50	66.00 99/150
retention only (no recovery search)	25.74	32.74 55/168	50.67	59.33 89/150

bottom one. Estimating g only over probes that left the recorded path keeps the score usable at depth, where re-tracing reaches 0.39–0.57; an ungated score correlates at only 0.36–0.49 (Appendix E).

4.5 Ablation and Where the Gain Comes From

Among the recovery-point and search rules, removing state localization costs the most. In Table 2, replacing Verified State Localization by recovery points at geometric depths $\{L/3, L/2, 2L/3\}$ costs the most among the recovery-point and search rules, 2.43 and 3.55 Avg@4 points: geometric placement often lands after the damaging action. The probes and the marginal-value rule of Eq. 5 are rules for scarce attempts: at three attempts per task they are worth 3.12 and 2.23 Avg@4 points on AppWorld and 2.84 and 2.34 on BFCL-V3, at the default of nine about one point, and under 0.6 at twelve, when almost every recovery point is tried anyway (Appendix Table A24). Restoring the environment state without the recorded context costs 1.14 and 0.72 points.

Replay decides Avg@4; the search decides Pass@4. Removing verified replay leaves Pass@4 unchanged, because a task still counts once when first solved, and costs 15.27 and 18.44 Avg@4 points; exposing the banked programs as callable functions, as a skill library does, recovers only part of that gap (22.02 and 51.50), because whether a program runs is still regenerated every pass. A text summary hurts both metrics (16.22 / 38.10 and 41.50 / 66.00), and pruning shortens programs from 11.8 to 6.9 actions on AppWorld at the cost of at most one task (sweeps: Appendix I).

Retention alone matches the prior state of the art; all of Pass@4 comes from the search. With the recovery search off (last row), Avg@4 still rises 10.42–13.50 points over No Memory on all eight settings, matching ReMe (dynamic) on AppWorld and passing it on BFCL-V3 (Appendix Figure A7), while Pass@4 equals No Memory's exactly. The search adds a further 6.33–18.60 Avg@4 points and all of the 7.33–24.41 Pass@4 gain; 27.50–57.27% of the tasks TaP solves are first solved from a recovery point (Appendix F).

5 Conclusion

We presented TaP, a training-free framework that stores each attempt of an LLM agent as an executable program and reuses it by executing it. Because a program restores a state rather than describing it, a new attempt can start from a verified recovery point inside a failed attempt, found by Verified State Localization and chosen by Probe-Guided Attempt Allocation, and a solved task is replayed in every later pass without a model call: the executed artefact, not the text written about it, carries an attempt to the next one.

Limitations. TaP needs an environment that resets and replays deterministically, as sandboxed applications, tools and code do and live services with side effects do not, and a checker that reports partial progress; with a success-only checker the recovery search loses its signal while verified replay still applies.

Reproducibility Statement

Appendix A lists the benchmarks, splits, protocol, every hyper-parameter and the runs behind each number; Appendix B counts every interaction. The code of TaP for both benchmarks, the run scripts, the banked programs of the reference runs and the per-run task counts behind every three-run number will be released.

References

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution, 2025. URL https://arxiv.org/abs/2512.10696.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. URL https://arxiv.org/abs/2507.06261.

DeepSeek-AI. Deepseek-v3 technical report, 2024. URL https://arxiv.org/abs/2412.19437.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, February 2021. ISSN 1476-4687. doi: 10.1038/s41586-020-03157-9. URL http://dx.doi.org/10.1038/s41586-020-03157-9.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory, 2025. URL https://arxiv.org/abs/2508.06433.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. In *arXiv preprint arXiv:2407.01476*, 2024.

LangChain. Langmem: Long-term memory for langgraph agents. https://github.com/langchain-ai/langmem, 2025.

Cong Lu, Shengran Hu, and Jeff Clune. Intelligent Go-Explore: Standing on the shoulders of giant foundation models. In *International Conference on Learning Representations (ICLR)*, 2025.

Mistral AI. Mistral-small-3.2-24b-instruct-2506. https://huggingface.co/mistralai/Mistral-Small-3.2-24B-Instruct-2506, 2025.

OpenAI. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

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

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.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Liyan Tang, Cyrus Rashtchian, Chun-Sung Ferng, Andrew Tomkins, Da-Cheng Juan, and Tu Vu. Wikiskill: Compiling agent experience into persistent knowledge for skill evolution, 2026. URL https://arxiv.org/abs/2608.27454.

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, 2024. URL https://arxiv.org/abs/2407.18901.

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, 2023a. URL https://arxiv.org/abs/2305.16291.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), March 2024a. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL http://dx.doi.org/10.1007/s11704-024-40231-1.

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

Zhiruo Wang, Daniel Fried, and Graham Neubig. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks, 2024b. URL https://arxiv.org/abs/2401.12869.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024c. URL https://arxiv.org/abs/2409.07429.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023a. URL https://arxiv.org/abs/2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023b. URL https://arxiv.org/abs/2210.03629.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners, 2024. URL https://arxiv.org/abs/2308.10144.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. In *arXiv preprint arXiv:2504.07079*, 2025.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control, 2024. URL https://arxiv.org/abs/2306.07863.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models, 2024. URL https://arxiv.org/abs/2310.04406.

AI USE STATEMENT

Generative AI tools were used only to assist with writing: polishing the grammar, wording and flow of text drafted by the authors. They were not used to generate synthetic data, to develop the theoretical model or conceptual framework, to formulate or prove mathematical claims, to propose or refine hypotheses, to design or give feedback on the methodology or experiments, to implement the method, to clean or reformat data, or to analyze or interpret results. The authors have reviewed all AI-assisted text and take full responsibility for the contents of this paper.

A Implementation Details

Benchmarks. AppWorld (Trivedi et al., 2024) tasks are day-to-day requests over a suite of simulated apps that the agent completes by writing and executing Python code against the apps' APIs; a task is solved when every check of its evaluator passes, and the partial score is the fraction of checks passed. We use the `test_normal` split (168 tasks). BFCL-V3 (Patil et al., 2025) `multi_turn_base` (150 tasks) gives the agent a sequence of user turns to satisfy with function calls; the checker scores every turn, so the progress profile of Section 3.2 is the vector of passed turns and a recovery point is a turn boundary. Both environments are reset to the task's initial state before every attempt, and executing the same action prefix after a reset reaches the same state, which is the assumption of Section 3.1.

Protocol. All methods follow the ReMe protocol (Cao et al., 2025): four evaluation passes over the task set, at most 30 interactions per attempt, sampling temperature 0.9, Qwen3 thinking mode disabled on AppWorld and enabled on BFCL-V3, Avg@4 and Pass@4 as metrics. Memory baselines populate their pools on the training split before the first pass, exactly as ReMe does; the published rows of Table 1 are taken from ReMe, and the rows we run (Reflexion, Memp, WikiSkill on every backbone; all methods on Mistral-Small-3.2-24B) use the official implementations of those methods ported to the same harness. TaP uses no training split and starts with an empty bank.

Evaluator use. Evaluator verdicts are used only to select frontier trajectories, to compute progress profiles, to label probes and full attempts, and to verify pruned programs. No evaluator output is ever placed in the model's context.

Hyper-parameters. Table A1 lists the configuration used for every result in the paper unless a row of an ablation or sweep says otherwise.

Table A1: Configuration of TaP used throughout the paper.

Setting	Value
Frontier trajectories per unsolved task	4; highest partial score, longer trajectory as tie-break
Recovery points probed per task	at most 4, highest verified progress first
Probes per recovery choice (d, T), m	2
Actions per probe, k	5
Temperature set for probes and full attempts	$\{0.7, 0.9, 1.1\}$ on AppWorld; $\{0.9, 1.1, 1.3\}$ on BFCL-V3
Temperature of fresh attempts	0.9 (protocol)
Prior strength c_0	1
Full attempts per unsolved task per round	9 (one global pool, spent by Eq. 5)
Deletion rule	≥ 4 deviating, non-improving attempts across ≥ 2 temperatures
Interactions per attempt	30
Evaluation passes	4

Closed models. GPT-4o, DeepSeek-V3 and Gemini 2.5 Pro are accessed through their APIs. The only protocol knob they need to expose is the temperature; where a seed is unavailable, the fresh arm draws ordinary samples. Everything else is identical to the open-backbone runs.

Runs. Table 1, the closed-model results and the within-attempt search baselines are means and sample standard deviations of three independent runs, and the per-run task counts (tasks solved in each pass and tasks solved at least once, per run) behind every three-run number of a method we run are released with the code. Per-pass curves, ablations, sweeps and all mechanism statistics are computed on the reference run of TaP with integer task counts; the baseline curves in Figures 3 and A1 are means of three runs, and the No Memory curve is our run with the seeds of the reference run. Table 1 keeps the published No Memory, A-Mem, LangMem and ReMe numbers for the Qwen3 backbones. The reference run is one of the three runs behind the TaP row of Table 1 and lies within 0.3 points of the three-run mean on both metrics. Our own No Memory runs agree with the published rows to within a point except on BFCL-V3 with Qwen3-14B, where the fresh rate of our serving stack is 44.33 Avg@4 against the published 48.66 (Table A14); every method we run there, TaP included, starts from that fresh rate.

Algorithm details. One search round runs between two evaluation passes and proceeds as follows; the settings are those of Table A1.

1. **Frontier and recovery points.** For each unsolved task, the four failed trajectories with the highest partial score (longer first on ties) form the frontier. Every new frontier trajectory is replayed once and evaluated at every prefix, which gives the profile $e_1, \ldots, e_L$ and the candidate depths of Eq. 2; the four candidates with the highest e_d (shallower first on ties) are the task's recovery points. A task whose frontier has no verified progress ($e_d = 0$ for every d, 24–42% of the pass-1 failures in Table A10) has no recovery point and receives no search attempts in the round; its next evaluation pass is an ordinary fresh attempt, and any progress that attempt makes enters the frontier.
2. **Probes.** Every recovery choice (d, T) receives $m = 2$ probes of at most $k = 5$ actions. A probe that generates k' actions is compared with the recorded trajectory at depth $d + k'$: $D = 1$ if its observation sequence differs from the recorded one anywhere in $(d, d + k']$, and, for a deviating probe, $I = 1$ if its progress after k' actions exceeds the recorded $e_{d+k'}$ (e_L if $d + k' > L$). Probes that produce no observation are discarded. $\rho(d, T)$ and $g(d)$ are frequency estimates with a Laplace prior, $\hat{\rho} = (n_D + 1)/(n + 2)$ per temperature and $\hat{g} = (n_I + 1)/(n_D + 2)$ pooled over temperatures, so $q = \hat{\rho}\hat{g}$ lies strictly inside $(0, 1)$ and both parameters of $\mathrm{Beta}(c_0 q, c_0(1 - q))$ are positive.
3. **Allocation.** The pool holds nine full attempts per unsolved task. One sample $\widehat{\pi}_u$ is drawn per choice, and the whole pool is filled by Eq. 5 before any attempt runs, ϕ_i being updated after every pick. The scheduled attempts then execute; a full attempt is labelled by its success y and by D and I against the final state of the recorded trajectory. The Beta posterior of a choice is updated with y only ($\alpha \leftarrow \alpha + y$, $\beta \leftarrow \beta + 1 - y$), never with progress. When an attempt solves its task, the task's remaining scheduled attempts are cancelled and their budget returns to the pool of the next round.
4. **Updating recovery points.** After an attempt from depth d with $D = 1$ and $I = 1$, every shallower recovery point $d' < d$ of the same trajectory has its prior mean raised to $\max\{q(d', T), \frac{1}{2}[q(d', T) + \bar{\pi}_d]\}$, where $\bar{\pi}_d$ is the posterior mean of the improving choice; the prior strength c_0 and the observation counts of d' are unchanged. The deletion rule of Table A1 removes a point and all deeper candidates of the same trajectory.
5. **Pruning.** The successful sequence C is shortened by deleting blocks of consecutive actions, with block size halving from $\lceil |C|/2 \rceil$ to 2 and blocks tried front to back, and then single actions back to front; a deletion is kept only if re-execution in a reset environment passes every check of $E(\tau)$ (Eq. 6), and an execution error counts as a failed check. The action that completes the task is never deleted. Pruning stops after 200 re-executions, and if the final program fails re-execution, the raw sequence C is banked.

B Interaction Accounting

Table A2 declares every interaction TaP makes beyond the four evaluation attempts, per task and over the whole protocol. BFCL-V3 evaluates turns, so prefix evaluation there is free. A replayed solved task makes no model call (Table A3), so the evaluation passes get cheaper as the bank fills and the additional interactions are spent in the search phase, where they are dominated by the full attempts. The evaluation passes themselves cost 17.7–51.7 further model calls per task: the fresh attempts at the tasks not yet banked (Table A17 gives the bank at the start of each pass), at the per-attempt cost of a No Memory pass. Table A4 gives each baseline the same total number of model calls per task, evaluation passes and search phase together, in the form of independent passes, and compares both Avg@K and Pass@K over those K passes. Every extended pass is a fresh sample with a new seed. The number of tasks a restart method solves for the first time decays geometrically from pass to pass: No Memory gains 2.58–6.45 Pass points from pass 4 to the matched K and ReMe (dynamic) 0.67–3.57, and TaP stays first on both metrics in every row. The passes behind these columns are in Appendix C (Tables A7 and A8).

Table A2: Additional test-time interactions per task over the whole four-pass protocol.

Backbone	Benchmark	probe episodes	full attempts	offline prefix evaluations	environment steps	model calls
Qwen3-8B	AppWorld	11.85	18.86	40.55	695.3	376.29
Qwen3-8B	BFCL-V3	7.65	11.64	0.0	367.14	196.13
Qwen3-14B	AppWorld	9.43	14.2	29.84	498.39	265.89
Qwen3-14B	BFCL-V3	6.34	9.12	0.0	281.44	147.96
Qwen3-32B	AppWorld	8.92	13.55	28.69	445.31	234.52
Qwen3-32B	BFCL-V3	5.57	8.46	0.0	247.94	130.43
Mistral-Small-3.2-24B	AppWorld	11.07	17.3	36.99	620.39	335.34
Mistral-Small-3.2-24B	BFCL-V3	7.27	10.98	0.0	340.98	181.94

Table A3: What a solved task costs in every later pass (Qwen3-8B): model calls and wall-clock latency per task.

	AppWorld		BFCL-V3	
Treatment of a solved task	model calls	latency (s)	model calls	latency (s)
regenerate the task (any text-based memory)	18.1	79.1	11.6	42.0
TaP: replay the stored program	0	7.3	0	4.1

Table A4: Budget-matched comparison. Every baseline is given K independent passes so that its model calls per task match the total TaP spends over the four evaluation passes and the search phase (the fresh attempts of the evaluation passes plus Table A2); Avg@K is the mean over those K passes and Pass@K the fraction of tasks solved in at least one of them.

		TaP, $K = 4$			No Memory, matched				ReMe (dynamic), matched			
Backbone	Benchmark	calls	Avg@4	Pass@4	K	calls	Avg@K	Pass@K	K	calls	Avg@K	Pass@K
Qwen3-8B	AppWorld	428.01	**34.47**	**48.02**	24	434.4	15.03	35.32	20	430.0	28.12	44.84
Qwen3-14B	AppWorld	302.45	**51.69**	**65.48**	18	298.8	22.59	46.83	15	300.0	38.69	55.95
Qwen3-32B	AppWorld	266.63	**54.66**	**70.04**	18	273.6	28.87	55.36	14	260.4	47.29	67.06
Mistral-Small-3.2-24B	AppWorld	381.01	**40.43**	**53.97**	22	382.8	17.42	41.67	18	374.4	31.13	48.21
Qwen3-8B	BFCL-V3	219.25	**60.11**	**70.44**	19	220.4	39.61	65.78	15	225.0	48.22	68.67
Qwen3-14B	BFCL-V3	166.16	**69.72**	**81.11**	15	165.0	44.31	74.44	12	172.8	58.48	75.11
Qwen3-32B	BFCL-V3	148.14	**71.44**	**80.00**	14	144.2	54.29	77.78	11	150.7	59.60	77.11
Mistral-Small-3.2-24B	BFCL-V3	203.79	**62.39**	**71.78**	18	203.4	42.04	63.56	14	205.8	50.29	70.00

C PER-PASS CURVES ON ALL SETTINGS AND BEYOND FOUR PASSES

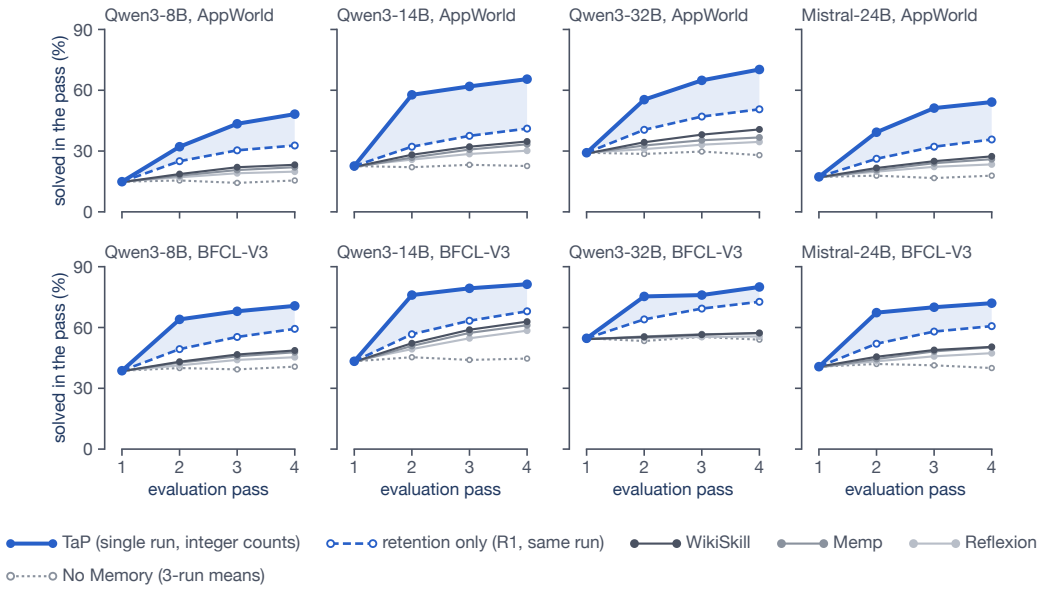


Figure A1: Success rate per evaluation pass for every backbone (columns) on AppWorld (top) and BFCL-V3 (bottom), encoding as in Figure 3. The separation opens after pass 1 on every setting, and TaP is non-decreasing on all eight. Per-pass values are in Table A5.

Table A5: Success rate (%) in each pass for every setting (the curves of Figure A1). TaP is the reference run; No Memory is our run with the same seeds, so its pass 1 is the same fresh attempt; Reflexion, Memp and WikiSkill are means of three runs in the same harness, with the four-pass mean of Table 1.

Benchmark	Backbone / method	pass 1	pass 2	pass 3	pass 4	Avg@4
AppWorld	Qwen3-8B, TaP	14.88	32.14	43.45	48.21	34.67
	No Memory	14.88	15.48	14.29	15.48	15.03
	Reflexion	15.08	17.06	19.05	19.84	17.76
	Memp	14.68	17.86	20.63	22.02	18.8
	WikiSkill	14.68	18.65	22.02	23.21	19.64
AppWorld	Qwen3-14B, TaP	22.62	57.74	61.9	65.48	51.93
	No Memory	22.62	22.02	23.21	22.62	22.62
	Reflexion	22.62	25.79	28.57	30.16	26.79
	Memp	22.02	26.79	30.75	33.33	28.22
	WikiSkill	22.22	28.17	32.14	34.72	29.32
AppWorld	Qwen3-32B, TaP	29.17	55.36	64.88	70.24	54.91
	No Memory	29.17	28.57	29.76	27.98	28.87
	Reflexion	29.17	30.95	33.13	34.52	31.94
	Memp	28.97	32.74	35.32	36.71	33.43
	WikiSkill	28.77	34.33	38.1	40.67	35.47
AppWorld	Mistral-Small-3.2-24B, TaP	17.26	39.29	51.19	54.17	40.48
	No Memory	17.26	17.86	16.67	17.86	17.41
	Reflexion	17.26	19.84	22.22	23.41	20.68
	Memp	16.87	20.63	24.01	25.99	21.88
	WikiSkill	17.06	21.63	25.0	27.38	22.77
BFCL-V3	Qwen3-8B, TaP	38.67	64.0	68.0	70.67	60.33
	No Memory	38.67	40.0	39.33	40.67	39.67
	Reflexion	38.67	41.33	44.0	45.33	42.33
	Memp	38.44	42.67	45.78	47.78	43.67
	WikiSkill	38.44	43.11	46.67	48.67	44.22
BFCL-V3	Qwen3-14B, TaP	43.33	76.0	79.33	81.33	70.0
	No Memory	43.33	45.33	44.0	44.67	44.33
	Reflexion	43.56	49.33	54.67	58.44	51.5
	Memp	42.89	50.89	57.33	61.11	53.06
	WikiSkill	43.11	52.22	58.89	62.89	54.28
BFCL-V3	Qwen3-32B, TaP	54.67	75.33	76.0	80.0	71.5
	No Memory	54.67	53.33	55.33	54.0	54.33
	Reflexion	54.67	54.89	55.33	55.56	55.11
	Memp	54.22	55.56	56.67	57.11	55.89
	WikiSkill	54.22	55.33	56.44	57.33	55.83
BFCL-V3	Mistral-Small-3.2-24B, TaP	40.67	67.33	70.0	72.0	62.5
	No Memory	40.67	42.0	41.33	40.0	41.0
	Reflexion	40.67	43.33	45.78	47.33	44.28
	Memp	40.22	44.44	48.22	50.22	45.78
	WikiSkill	40.67	45.56	48.89	50.44	46.39

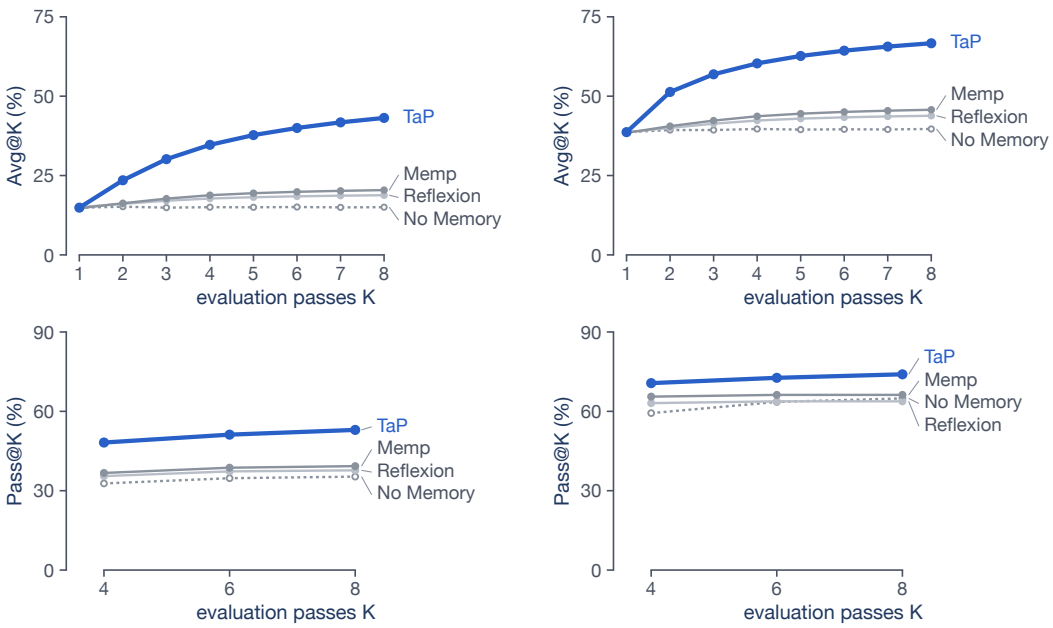


Figure A2: Beyond four passes (Qwen3-8B). Top: Avg@K for $K = 1$ to 8 evaluation passes on AppWorld (left) and BFCL-V3 (right). TaP keeps rising through $K = 8$ because every solved task keeps contributing without being re-solved, while the baselines flatten once their pools stop filling. Bottom: Pass@K at $K = 4$, 6 and 8; $K = 4$ is the reference run. Values in Table A6.

Table A6: Number of passes K (Qwen3-8B). The four-pass protocol is extended to six and eight passes with the same per-round search; TaP is the reference run with integer counts, the baselines are our three-run means at every K, so their K=4 values are our runs rather than the published rows of Table 1. New first solves per additional pass shrink as the unsolved pool concentrates on tasks that never reach a verified state, and Avg@K rises faster than Pass@K because every already-solved task contributes a full point in every later pass. Every extended pass is a fresh sample with a new seed. The Pass@K of the baselines flattens at two speeds: No Memory keeps finding tasks at the decaying rate of its own passes 2–4, Reflexion and Memp re-solve what they solved (Figure 4b); Tables A7 and A8 follow No Memory and ReMe (dynamic) to the matched K of Table A4.

(a) AppWorld

K	TaP Avg@K	TaP Pass@K	No Memory Pass@K	Reflexion Pass@K	Memp Pass@K
4	34.67	48.21 (81/168)	32.74	35.52	36.71
6	39.98	51.19 (86/168)	34.72	37.30	38.69
8	43.15	52.98 (89/168)	35.32	37.70	39.29

(b) BFCL-V3

K	TaP Avg@K	TaP Pass@K	No Memory Pass@K	Reflexion Pass@K	Memp Pass@K
4	60.33	70.67 (106/150)	59.33	63.11	65.56
6	64.33	72.67 (109/150)	63.56	63.78	66.22
8	66.67	74.00 (111/150)	64.89	63.78	66.22

Extended passes to the matched budget. Tables A7 and A8 give the passes behind the Pass@K columns of Table A4. Every pass beyond the fourth is a fresh sample with a new seed; no sample is reused, and the memory of ReMe (dynamic) keeps updating as in its first four passes. The number of tasks a restart method solves for the first time decays geometrically from pass to pass, at a ratio that the method's own passes 2–4 set: 0.48–0.63 for No Memory (its passes 2–4 are the run of Table A14), 0.46 on AppWorld and 0.28 on BFCL-V3 for the text memories, the ratio at which Reflexion and Memp of Table A6 close their unions and which ReMe (dynamic) follows. The two speeds are the two regression rates of Figure 4b: a fresh attempt re-solves a solved task 15.4–54.3% of the time, so its successes are spread over more tasks at lower per-task rates and later samples

keep finding some, whereas a text memory re-solves 42.1–77.9% of the time, so its successes are concentrated and its union closes within a few passes. No Memory gains 2.58–6.45 Pass points from pass 4 to the matched K and ReMe (dynamic) 0.67–3.57; on BFCL-V3 with Qwen3-14B and Qwen3-32B, fifteen and fourteen fresh passes reach the Pass@K of twelve and eleven ReMe passes (74.44 against 75.11 and 77.78 against 77.11), since the memory raises the per-attempt rate on tasks that a fresh attempt can also reach. TaP stays 2.22–18.65 points above No Memory and 1.77–9.53 above ReMe (dynamic) at the matched budget.

Table A7: Tasks solved for the first time at each pass (Qwen3-8B), up to the matched K of Table A4 (underlined; means of three runs, fresh seeds for every extended pass). One of the three No Memory runs is the run of Table A14; the union of the three at pass 4 is 32.74 and 59.33, the No Memory row of Table 1 being the published one.

		tasks solved for the first time at pass t																						
Benchmark	Method	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
AppWorld	No Memory	25	17.3	8.3	4.3	2.3	1.0	0.7	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AppWorld	ReMe (dynamic)	32.3	23.0	10.3	5.0	2.7	1.0	0.7	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	–	–	–	–
BFCL-V3	No Memory	58	16.0	9.0	6.0	4.0	2.3	1.3	0.7	0.7	0.3	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	–	–	–	–	–
BFCL-V3	ReMe (dynamic)	59.7	31.3	9.0	2.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	–	–	–	–	–	–	–	–	–

Table A8: Pass@K of No Memory and ReMe (dynamic) at $K = 4$, 6, 8, 12 and the matched K of Table A4 (in parentheses), all eight settings, means of three runs; the $K = 4$ column is our runs and lies within a point of Table 1.

		No Memory					ReMe (dynamic)				
Backbone	Benchmark	K=4	6	8	12	matched K	K=4	6	8	12	matched K
Qwen3-8B	AppWorld	32.74	34.72	35.32	35.32	35.32 (24)	42.06	44.25	44.64	44.84	44.84 (20)
Qwen3-14B	AppWorld	41.07	44.84	46.03	46.83	46.83 (18)	52.98	55.36	55.95	55.95	55.95 (15)
Qwen3-32B	AppWorld	50.60	53.97	54.96	55.36	55.36 (18)	63.49	66.27	66.87	67.06	67.06 (14)
Mistral-Small-3.2-24B	AppWorld	35.32	39.29	40.67	41.47	41.67 (22)	45.44	47.62	48.02	48.21	48.21 (18)
Qwen3-8B	BFCL-V3	59.33	63.56	64.89	65.78	65.78 (19)	68.00	68.67	68.67	68.67	68.67 (15)
Qwen3-14B	BFCL-V3	68.00	72.44	73.78	74.44	74.44 (15)	74.44	75.11	75.11	75.11	75.11 (12)
Qwen3-32B	BFCL-V3	72.67	76.00	77.11	77.78	77.78 (14)	76.44	77.11	77.11	77.11	77.11 (11)
Mistral-Small-3.2-24B	BFCL-V3	60.89	63.11	63.56	63.56	63.56 (18)	69.33	70.00	70.00	70.00	70.00 (14)

Table A9: Within-attempt search under the same protocol (Qwen3-8B; the search agents are means of three runs, TaP is the reference run), per pass. The interaction budget per pass of the two search agents is matched to the total TaP spends per pass, search included. Both raise the pass-1 rate over No Memory and flatten from pass 2, because each attempt rebuilds its tree from the initial state.

Method	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
AppWorld, Qwen3-8B						
Tree search inside the attempt (LATS-style)	52	51	53	51	30.80	41.67
Best-of-n branching inside the attempt (ToT-style)	49	48	50	49	29.17	39.29
TaP	25	54	73	81	34.67	48.21
BFCL-V3, Qwen3-8B						
Tree search inside the attempt (LATS-style)	86	87	86	86	57.50	67.33
Best-of-n branching inside the attempt (ToT-style)	84	84	84	84	56.00	65.33
TaP	58	96	102	106	60.33	70.67

D VERIFIED PROGRESS IN FAILED ATTEMPTS

Figure A3 and Table A10 summarise the pass-1 failures of the reference run. Between 58.04% and 76.47% of them have earned at least one check beyond the initial state; among those, the median has completed a third to a half of the task's checks and 26.51–48.08% have completed more than half. The fraction carrying progress grows with backbone size on both benchmarks. The mean number of recovery points per progressed trajectory grows on AppWorld (1.97 to 2.34 from Qwen3-8B to Qwen3-32B) and moves little on BFCL-V3 (1.85 to 2.08), where progress is quantised by turns, which bounds the number of plateaus a trajectory can have regardless of backbone.

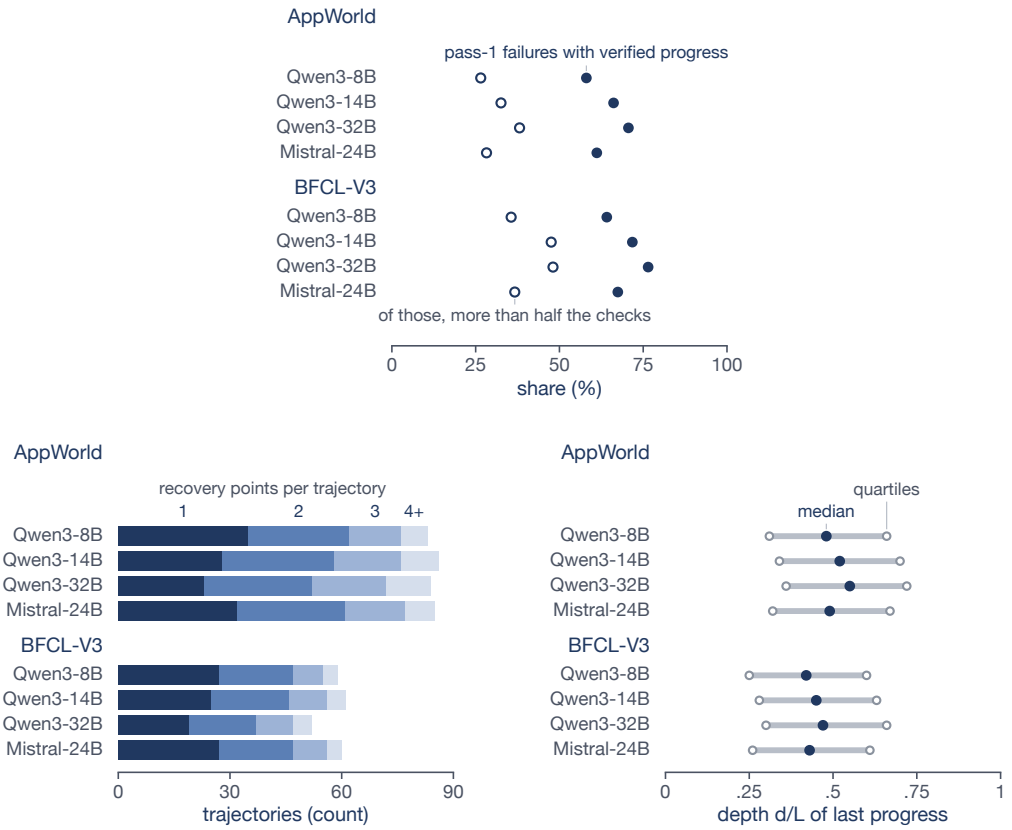


Figure A3: Verified progress in pass-1 failures. (a) Share of failed pass-1 trajectories that have earned at least one new check and, among those, the share that completed more than half of the task's checks. (b) Number of recovery points selected by Verified State Localization per progressed trajectory (counts). (c) Quartiles of the relative depth d/L of the last progress event: the median sits between $0.42L$ and $0.55L$ on every setting.

Table A10: Statistics of pass-1 failures: failed trajectories, those with verified progress (share in parentheses), the share of progressed failures that completed more than half of the checks, quartiles of the final partial score p among progressed failures, the number of trajectories with 1 / 2 / 3 / 4 or more recovery points, and quartiles of the relative depth of the last progress event.

Benchmark	Backbone	failed	with progress	>half (%)	p quartiles	rec. points 1/2/3/4+	d_{last}/L quartiles
AppWorld	Qwen3-8B	143	83 (58.04%)	26.51	0.17 / 0.33 / 0.5	35 / 27 / 14 / 7	0.31 / 0.48 / 0.66
AppWorld	Qwen3-14B	130	86 (66.15%)	32.56	0.2 / 0.4 / 0.57	28 / 30 / 18 / 10	0.34 / 0.52 / 0.7
AppWorld	Qwen3-32B	119	84 (70.59%)	38.1	0.25 / 0.43 / 0.6	23 / 29 / 20 / 12	0.36 / 0.55 / 0.72
AppWorld	Mistral-Small-3.2-24B	139	85 (61.15%)	28.24	0.17 / 0.36 / 0.5	32 / 29 / 16 / 8	0.32 / 0.49 / 0.67
BFCL-V3	Qwen3-8B	92	59 (64.13%)	35.59	0.25 / 0.4 / 0.6	27 / 20 / 8 / 4	0.25 / 0.42 / 0.6
BFCL-V3	Qwen3-14B	85	61 (71.76%)	47.54	0.33 / 0.5 / 0.67	25 / 21 / 10 / 5	0.28 / 0.45 / 0.63
BFCL-V3	Qwen3-32B	68	52 (76.47%)	48.08	0.33 / 0.5 / 0.67	19 / 18 / 10 / 5	0.3 / 0.47 / 0.66
BFCL-V3	Mistral-Small-3.2-24B	89	60 (67.42%)	36.67	0.25 / 0.4 / 0.6	27 / 20 / 9 / 4	0.26 / 0.43 / 0.61

E Recovery Depth, Observation Gating and Probe Calibration

Depth trade-off on every setting. Figure A4 repeats Figure 5a for all eight backbone–benchmark settings and Table A11 lists the values. ρ falls with depth at every temperature on every panel, g peaks in the $(0.6, 0.8]$ bin on every panel, and q peaks in $(0.4, 0.6]$ on six settings and in $(0.6, 0.8]$ on two (Qwen3-14B and Mistral-Small-3.2-24B on BFCL-V3). Larger backbones raise g at every depth while leaving ρ almost unchanged, so their peak sits slightly deeper. The temperature grid differs between benchmarks because the protocol uses a different default set on each (Table A1).

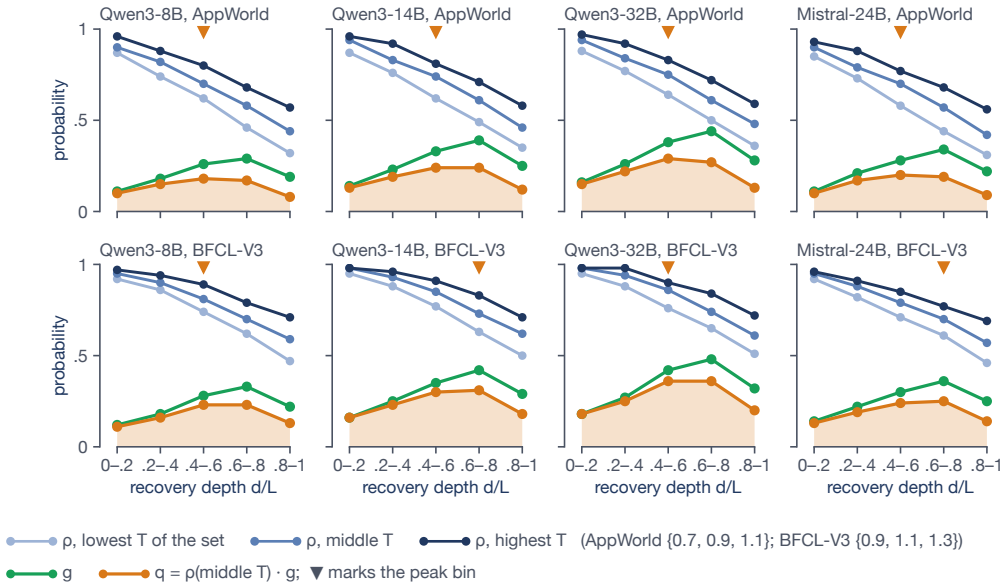


Figure A4: Recovery-depth trade-off on all eight backbone–benchmark settings, encoding as in Figure 5a; the temperature set differs by benchmark (legend), and q uses the middle temperature of each set. q peaks in an interior bin on every panel.

Table A11: Values behind Figure A4: probes binned by the relative depth of their recovery point (probes without an observation are discarded and counted in parentheses); n, ρ per temperature, g, and $q = \rho g$ at the middle temperature per bin, with the peak of q in bold.

Benchmark	Backbone	probes (disc.)	quantity	$(0, .2]$	$(.2, .4]$	$(.4, .6]$	$(.6, .8]$	$(.8, 1)$
AppWorld	Qwen3-8B	1969 (22)	n	277	545	608	405	134
			$\rho, T = 0.7$	0.87	0.74	0.62	0.46	0.32
			$\rho, T = 0.9$	0.9	0.82	0.7	0.58	0.44
			$\rho, T = 1.1$	0.96	0.88	0.8	0.68	0.57
			g	0.11	0.18	0.26	0.29	0.19
			$q = \rho g\ (T = 0.9)$	0.1	0.15	**0.18**	0.17	0.08
AppWorld	Qwen3-14B	1568 (17)	n	204	452	507	319	86
			$\rho, T = 0.7$	0.87	0.76	0.62	0.49	0.35
			$\rho, T = 0.9$	0.94	0.83	0.74	0.61	0.46
			$\rho, T = 1.1$	0.96	0.92	0.81	0.71	0.58
			g	0.14	0.23	0.33	0.39	0.25
			$q = \rho g\ (T = 0.9)$	0.13	0.19	**0.24**	0.24	0.12
AppWorld	Qwen3-32B	1482 (16)	n	180	439	497	297	69
			$\rho, T = 0.7$	0.88	0.77	0.64	0.5	0.36
			$\rho, T = 0.9$	0.94	0.84	0.75	0.61	0.48
			$\rho, T = 1.1$	0.97	0.92	0.83	0.72	0.59
			g	0.16	0.26	0.38	0.44	0.28
			$q = \rho g\ (T = 0.9)$	0.15	0.22	**0.29**	0.27	0.13
AppWorld	Mistral-Small-3.2-24B	1840 (20)	n	254	515	574	378	119
			$\rho, T = 0.7$	0.85	0.73	0.58	0.44	0.31
			$\rho, T = 0.9$	0.9	0.79	0.7	0.57	0.42
			$\rho, T = 1.1$	0.93	0.88	0.77	0.68	0.56
			g	0.11	0.21	0.28	0.34	0.22
			$q = \rho g\ (T = 0.9)$	0.1	0.17	**0.2**	0.19	0.09
BFCL-V3	Qwen3-8B	1134 (13)	n	176	296	326	235	101
			$\rho, T = 0.9$	0.92	0.86	0.74	0.62	0.47
			$\rho, T = 1.1$	0.95	0.9	0.81	0.7	0.59
			$\rho, T = 1.3$	0.97	0.94	0.89	0.79	0.71
			g	0.12	0.18	0.28	0.33	0.22
			$q = \rho g\ (T = 1.1)$	0.11	0.16	**0.23**	0.23	0.13
BFCL-V3	Qwen3-14B	941 (10)	n	139	253	280	195	74
			$\rho, T = 0.9$	0.95	0.88	0.77	0.63	0.5
			$\rho, T = 1.1$	0.98	0.93	0.85	0.73	0.62
			$\rho, T = 1.3$	0.98	0.96	0.91	0.83	0.71
			g	0.16	0.25	0.35	0.42	0.29
			$q = \rho g\ (T = 1.1)$	0.16	0.23	0.3	**0.31**	0.18
BFCL-V3	Qwen3-32B	827 (9)	n	118	227	252	171	59
			$\rho, T = 0.9$	0.95	0.88	0.76	0.65	0.51
			$\rho, T = 1.1$	0.98	0.94	0.86	0.74	0.61
			$\rho, T = 1.3$	0.98	0.98	0.9	0.84	0.72
			g	0.18	0.27	0.42	0.48	0.32
			$q = \rho g\ (T = 1.1)$	0.18	0.25	**0.36**	0.36	0.2
BFCL-V3	Mistral-Small-3.2-24B	1078 (12)	n	165	284	313	224	92
			$\rho, T = 0.9$	0.92	0.82	0.71	0.61	0.46
			$\rho, T = 1.1$	0.95	0.88	0.79	0.7	0.57
			$\rho, T = 1.3$	0.96	0.91	0.85	0.77	0.69
			g	0.14	0.22	0.3	0.36	0.25
			$q = \rho g\ (T = 1.1)$	0.13	0.19	0.24	**0.25**	0.14

Observation gating. Section 3.3 estimates g only over probes that left the recorded observation sequence. Figure A5 and Table A12 test this rule against an ungated score that counts a re-tracing probe as a failure of the state, on the same recovery choices as Figure 5b. Re-tracing rises from 0.03–0.11 in the shallowest bin to 0.39–0.57 in the deepest, so the ungated score charges the state for what is a property of the conditioning. The gated score predicts the realised success of full attempts with a Spearman correlation of 0.57–0.66 against 0.36–0.49 for the ungated score, its within-task rank agreement with realised success is 0.69–0.73 against 0.50–0.58, and 24.74–28.65% of the choices change rank between the two scores, almost all of them deep. The deviation label is taken from the observation sequence rather than from the action text because two different actions with the same effect produce the same observations and are correctly treated as a re-trace.

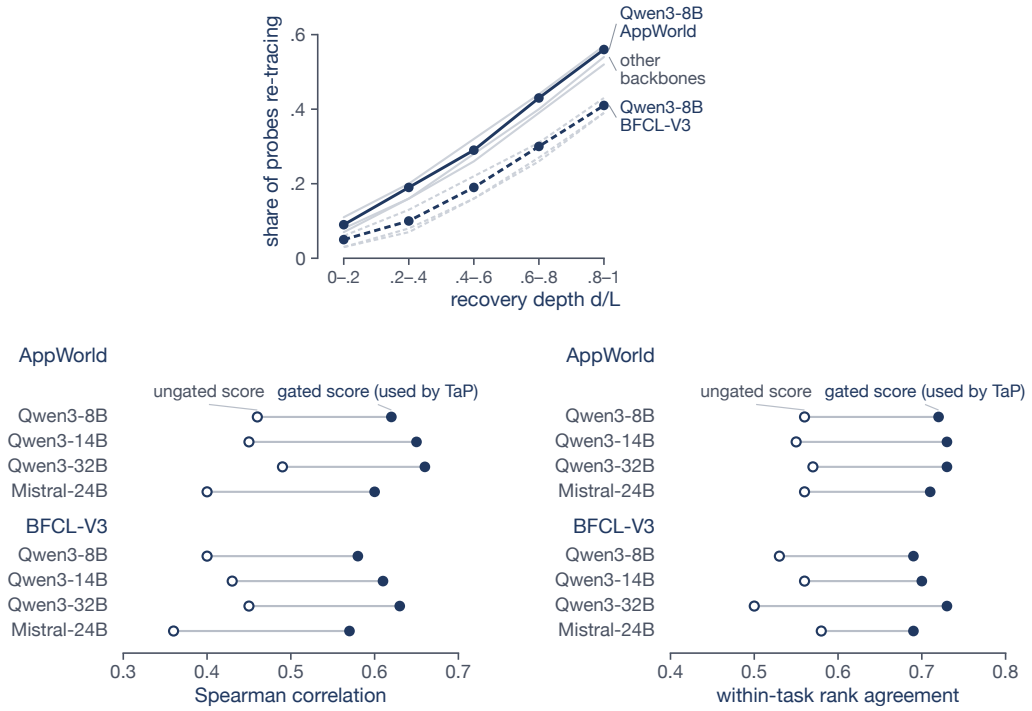


Figure A5: Observation gating. (a) Probability that a probe re-traces the recorded observation sequence, by depth bin (Qwen3-8B in ink, other backbones in grey; solid AppWorld, dashed BFCL-V3). (b) Spearman correlation between a choice's probe score and the realised success of its full attempts, for the gated score used by TaP and the ungated score. (c) Within-task rank agreement of the two scores with realised success.

Table A12: Gated against ungated probe score on the recovery choices that received full attempts. The calibration slope is the ratio of realised success to score; the gated slope is the factor by which q is optimistic in Figure 5b.

			Spearman		within-task rank agreement		calibration slope	
Benchmark	Backbone	choices	gated	ungated	gated	ungated	gated	ungated
AppWorld	Qwen3-8B	246	0.62	0.46	0.72	0.56	0.185	0.289
AppWorld	Qwen3-14B	198	0.65	0.45	0.73	0.55	0.255	0.398
AppWorld	Qwen3-32B	171	0.66	0.49	0.73	0.57	0.205	0.313
AppWorld	Mistral-Small-3.2-24B	232	0.6	0.4	0.71	0.56	0.2	0.31
BFCL-V3	Qwen3-8B	158	0.58	0.4	0.69	0.53	0.235	0.348
BFCL-V3	Qwen3-14B	126	0.61	0.43	0.7	0.56	0.33	0.487
BFCL-V3	Qwen3-32B	97	0.63	0.45	0.73	0.5	0.29	0.45
BFCL-V3	Mistral-Small-3.2-24B	149	0.57	0.36	0.69	0.58	0.255	0.393

Probe calibration on every backbone. Figure A6 adds the other three backbones to Figure 5b, and Table A13 lists the quintile values. Realised success is measured per full attempt, not per task: several attempts from the same choice can solve the same task in one round, and an attempt that succeeds after the task is already banked still counts as a successful attempt. The Beta posterior of a choice is updated only by full-attempt outcomes, so the uniform optimism of q is corrected within a round rather than accumulating.

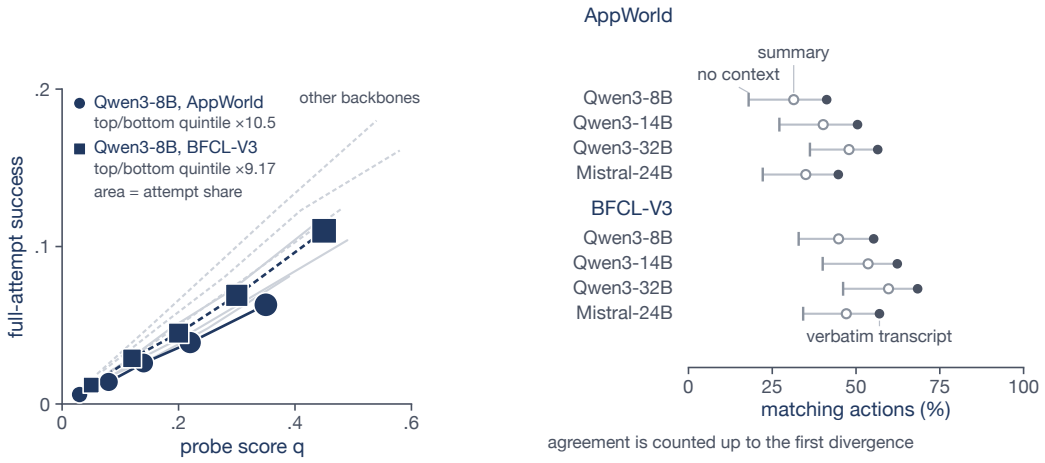
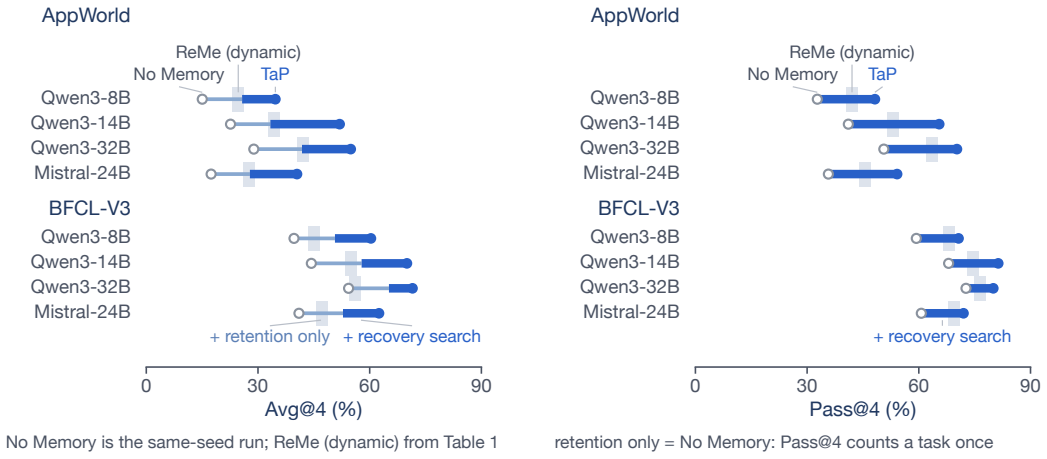


Figure A6: Left: probe calibration on all eight backbone–benchmark settings, encoding as in Figure 5b; grey curves are Qwen3-14B, Qwen3-32B and Mistral-Small-3.2-24B (solid AppWorld, dashed BFCL-V3). Every series is monotone in q. Right: action agreement for the regenerating forms of Figure 4a, the fraction of regenerated actions whose observation matches the recorded observation at the same step, counted up to the first divergence (values in Table A15).

Table A13: Probe score against realised full-attempt success by quintile of q (values of Figures 5b and A6): mean q, realised success rate of the full attempts, and the share of all full attempts spent on the quintile.

Benchmark	Backbone	choices	quantity	Q1	Q2	Q3	Q4	Q5
AppWorld	Qwen3-8B	246	mean q	0.03	0.08	0.14	0.22	0.35
			realised success	0.006	0.014	0.026	0.039	0.063
		Spearman 0.62	share of full attempts	0.117	0.173	0.187	0.244	0.279
AppWorld	Qwen3-14B	198	mean q	0.04	0.11	0.19	0.29	0.44
			realised success	0.011	0.027	0.049	0.072	0.116
		Spearman 0.65	share of full attempts	0.102	0.146	0.205	0.255	0.293
AppWorld	Qwen3-32B	171	mean q	0.05	0.13	0.22	0.33	0.49
			realised success	0.01	0.028	0.045	0.069	0.104
		Spearman 0.66	share of full attempts	0.108	0.147	0.202	0.262	0.28
AppWorld	Mistral-Small-3.2-24B	232	mean q	0.03	0.09	0.16	0.25	0.39
			realised success	0.006	0.019	0.03	0.048	0.081
		Spearman 0.6	share of full attempts	0.102	0.152	0.213	0.244	0.29
BFCL-V3	Qwen3-8B	158	mean q	0.05	0.12	0.2	0.3	0.45
			realised success	0.012	0.029	0.045	0.069	0.11
		Spearman 0.58	share of full attempts	0.104	0.158	0.189	0.242	0.307
BFCL-V3	Qwen3-14B	126	mean q	0.06	0.15	0.25	0.37	0.54
			realised success	0.019	0.049	0.083	0.124	0.18
		Spearman 0.61	share of full attempts	0.097	0.135	0.191	0.245	0.331
BFCL-V3	Qwen3-32B	97	mean q	0.07	0.17	0.28	0.41	0.58
			realised success	0.021	0.05	0.081	0.123	0.161
		Spearman 0.63	share of full attempts	0.107	0.141	0.188	0.26	0.304
BFCL-V3	Mistral-Small-3.2-24B	149	mean q	0.05	0.13	0.22	0.33	0.48
			realised success	0.013	0.032	0.055	0.084	0.124
		Spearman 0.57	share of full attempts	0.089	0.133	0.214	0.274	0.289

F RETENTION, RECOVERY SEARCH AND THE TWO STORES



(a) Avg@4 (b) Pass@4

Figure A7: Retention and recovery search contribute separately. Each row decomposes TaP into what verified replay alone achieves (retention only) and what the recovery search adds, against the No Memory run with the same seeds; the pale band marks ReMe (dynamic) from Table 1. (a) Avg@4: the thin segment is retention, worth 10.42–13.50 points, the thick segment the search, worth a further 6.33–18.60. (b) Pass@4: retention adds nothing by construction, because a task counts once in the pass in which it is first solved, so retention only coincides with No Memory on every row and the entire gain of 7.33–24.41 points comes from the recovery search.

Table A14 gives the numbers behind Figure A7 together with the number of tasks first solved in each of the three search rounds. Retention only is the multi-component removal of Section 4.5: verified replay without recovery points, probes or allocation. Its Pass@4 equals the No Memory Pass@4 of the same run on every setting. The fresh re-solve rate of tasks that remain unsolved decays quickly, from 10.79–23.53% in the first retry round to 3.42–12.73% in the third, which is why retention flattens after pass 2 in Figure 3 while the full method keeps rising. Table A15 and Table A16 give the values behind Figure 4.

Table A14: Decomposition of the two metrics (reference run). No Memory is the run with the same seeds; TaP is the reference run, one of the three runs behind Table 1. The last two columns give the tasks first solved from a recovery point in search rounds 1 / 2 / 3 and their share of all tasks TaP solves.

		Avg@4			Pass@4			first solves	from rec.
Benchmark	Backbone	No Mem.	retention	TaP	No Mem.	retention	TaP	round 1/2/3	point (%)
AppWorld	Qwen3-8B	15.03	25.74	**34.67**	32.74	32.74	**48.21**	24 / 11 / 5	49.38
AppWorld	Qwen3-14B	22.62	33.33	**51.93**	41.07	41.07	**65.48**	55 / 4 / 4	57.27
AppWorld	Qwen3-32B	28.87	41.82	**54.91**	50.6	50.6	**70.24**	42 / 16 / 8	55.93
AppWorld	Mistral-Small-3.2-24B	17.41	27.83	**40.48**	35.71	35.71	**54.17**	31 / 15 / 4	54.95
BFCL-V3	Qwen3-8B	39.67	50.67	**60.33**	59.33	59.33	**70.67**	37 / 5 / 3	42.45
BFCL-V3	Qwen3-14B	44.33	57.83	**70.0**	68.0	68.0	**81.33**	48 / 4 / 2	44.26
BFCL-V3	Qwen3-32B	54.33	65.17	**71.5**	72.67	72.67	**80.0**	27 / 1 / 5	27.5
BFCL-V3	Mistral-Small-3.2-24B	41.0	52.83	**62.5**	60.67	60.67	**72.0**	38 / 3 / 2	39.81

Table A15: The same successful experience in four forms (values of Figure 4a): tasks solved again (%) and action agreement (%) for the three regenerating forms; program replay re-solves 100% on every row.

			summary		verbatim transcript		no context	
Benchmark	Backbone	solved tasks	solved again	agreement	solved again	agreement	solved again	agreement
AppWorld	Qwen3-8B	81	25.31	31.4	34.88	41.24	31.17	17.95
AppWorld	Qwen3-14B	110	27.73	40.2	39.32	50.44	34.55	27.12
AppWorld	Qwen3-32B	118	33.47	47.9	45.55	56.47	41.1	36.24
AppWorld	Mistral-Small-3.2-24B	91	26.37	35.0	36.81	44.71	32.14	22.16
BFCL-V3	Qwen3-8B	106	47.17	44.8	63.21	55.26	56.13	32.88
BFCL-V3	Qwen3-14B	122	45.49	53.6	62.5	62.31	54.51	40.05
BFCL-V3	Qwen3-32B	120	55.0	59.7	77.5	68.38	67.92	46.14
BFCL-V3	Mistral-Small-3.2-24B	108	47.22	47.1	63.89	56.99	56.94	34.22

Table A16: Regression rate (%): solved tasks that fail again in the next pass, pooled over the three transitions (values of Figure 4b).

Benchmark	Backbone	No Memory	Reflexion	Memp	WikiSkill	TaP
AppWorld	Qwen3-8B	84.6	57.9	52.4	49.8	**0.0**
AppWorld	Qwen3-14B	76.8	49.5	45.1	42.7	**0.0**
AppWorld	Qwen3-32B	72.1	44.6	40.3	38.1	**0.0**
AppWorld	Mistral-Small-3.2-24B	82.3	55.0	50.2	47.4	**0.0**
BFCL-V3	Qwen3-8B	59.4	36.2	31.6	29.4	**0.0**
BFCL-V3	Qwen3-14B	51.3	30.8	26.9	24.8	**0.0**
BFCL-V3	Qwen3-32B	45.7	27.4	23.8	22.1	**0.0**
BFCL-V3	Mistral-Small-3.2-24B	57.9	34.9	30.1	28.0	**0.0**

The two stores. Figure A8 and Table A17 show what the solution bank and the state tree hold at the start of each pass. The bank grows in every pass and never shrinks, so the fraction of each pass that costs no model call rises monotonically and reaches 96.30–99.18% of the score by pass 4. The state tree moves the other way: it holds only unsolved tasks, so the recovery points it offers decrease after pass 2 and concentrate on progressively harder tasks, which is the mechanism behind the low conversion rates of Table A37. Programs are counted after verified pruning, so the bank grows in entries while each entry is shorter than the trajectory that produced it.

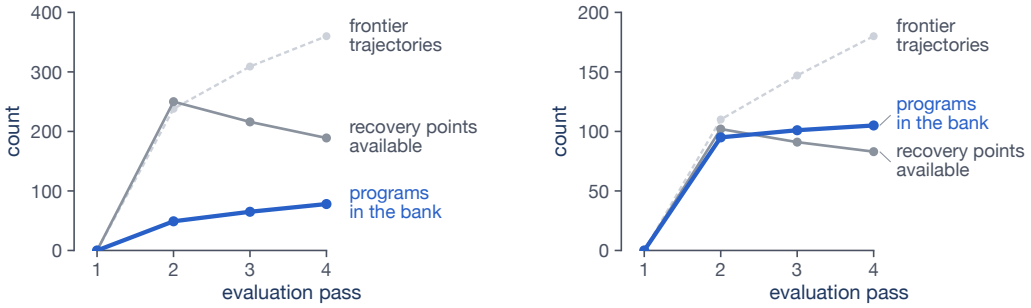
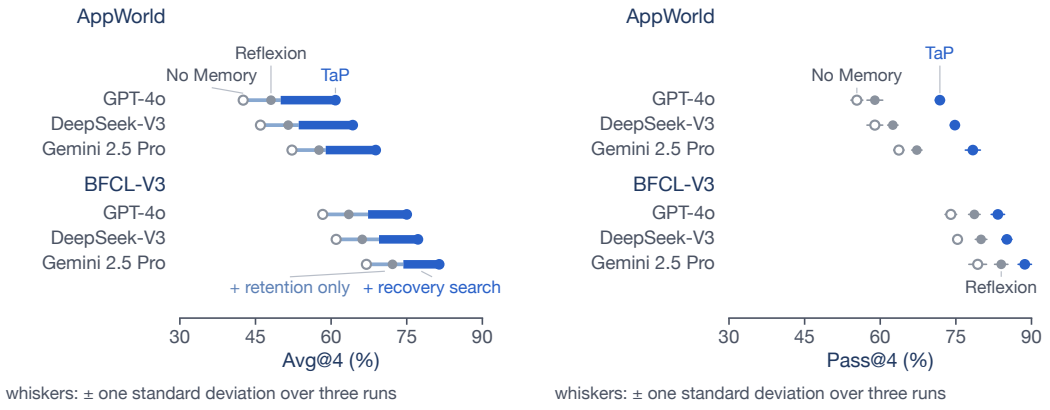


Figure A8: The two stores at the start of each pass (Qwen3-8B). The solution bank grows in every pass and never shrinks; the recovery points offered by the state tree, which holds only unsolved tasks, decrease after pass 2; frontier trajectories of unsolved tasks are shown for reference.

Table A17: The two stores on every setting: programs in the bank and recovery points available at the start of each pass, and the split of the pass-4 score between replayed (bank) and freshly solved tasks.

		programs in bank at pass				recovery points at pass				pass-4 score	
Benchmark	Backbone	1	2	3	4	1	2	3	4	bank / fresh	from bank (%)
AppWorld	Qwen3-8B	0	49	65	78	0	250	216	189	78 / 3	96.3
AppWorld	Qwen3-14B	0	93	101	108	0	158	141	126	108 / 2	98.18
AppWorld	Qwen3-32B	0	91	109	117	0	162	124	107	117 / 1	99.15
AppWorld	Mistral-Small-3.2-24B	0	60	81	90	0	227	183	164	90 / 1	98.9
BFCL-V3	Qwen3-8B	0	95	101	105	0	102	91	83	105 / 1	99.06
BFCL-V3	Qwen3-14B	0	113	118	121	0	68	59	54	121 / 1	99.18
BFCL-V3	Qwen3-32B	0	109	114	119	0	76	67	57	119 / 1	99.17
BFCL-V3	Mistral-Small-3.2-24B	0	99	104	107	0	94	85	80	107 / 1	99.07

G Closed Models and Cross-Model Transfer



(a) Avg@4 (b) Pass@4

Figure A9: The method applied unchanged to API-only backbones. GPT-4o, DeepSeek-V3 and Gemini 2.5 Pro under the protocol of Table 1; three independent runs, whiskers are one standard deviation. In (a) the thin segment is what verified replay alone contributes and the thick segment what the recovery search adds; Reflexion is the grey point. Values: Table A18.

Table A18: The method applied unchanged to API-only backbones (values of Figure A9); mean and standard deviation of three independent runs.

		No Memory		Reflexion		TaP	
Benchmark	Backbone	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4
AppWorld	GPT-4o	42.56±0.83	55.36±1.19	48.07±0.60	58.93±1.57	**60.86**±0.39	**71.83**±0.91
AppWorld	DeepSeek-V3	45.98±0.83	58.93±1.57	51.49±0.89	62.50±1.03	**64.29**±0.68	**74.80**±0.91
AppWorld	Gemini 2.5 Pro	52.23±0.65	63.69±1.03	57.59±0.60	67.26±1.03	**69.39**±0.48	**78.37**±1.50
BFCL-V3	GPT-4o	58.33±0.67	74.00±1.15	63.50±0.44	78.67±1.15	**75.00**±0.44	**83.33**±1.33
BFCL-V3	DeepSeek-V3	61.00±0.83	75.33±0.67	66.17±0.50	80.00±1.15	**77.22**±0.63	**85.11**±1.02
BFCL-V3	Gemini 2.5 Pro	67.00±0.58	79.33±1.76	72.17±0.60	84.00±1.33	**81.44**±0.54	**88.67**±1.33

Table A19: Decomposition of the closed-model runs: tasks solved per pass, what verified replay alone contributes, and the tasks first solved from a recovery point. As the backbone gets stronger, more tasks are solved on the first pass, retention has more to keep, and the residual pool the search works on becomes harder.

Benchmark	Backbone	solved per pass	retention-only Avg@4	tasks first solved from a recovery point	share of solved tasks (%)
AppWorld	GPT-4o	72 / 103 / 115 / 121	50.00	34	28.1
AppWorld	DeepSeek-V3	78 / 109 / 120 / 126	53.57	30	23.81
AppWorld	Gemini 2.5 Pro	88 / 117 / 127 / 132	58.93	27	20.45
BFCL-V3	GPT-4o	88 / 117 / 122 / 125	67.33	27	21.6
BFCL-V3	DeepSeek-V3	92 / 120 / 125 / 128	69.50	25	19.53
BFCL-V3	Gemini 2.5 Pro	101 / 126 / 130 / 133	74.33	20	15.04

Transfer of experience between models. A banked program is an executable artefact, so it can be handed to any other model: replaying it costs no model call and reproduces the verdict. A recovery point is weaker currency: it restores a state, but whoever continues from it still has to finish the task. Figure A10 and Table A20 measure both in four directions, including from a closed API model to an open 8B one. In the donor-programs arm the receiver makes no fresh attempts of its own: its four passes replay the donor's bank, and its search continues only from the recovery points of the donor's frontier, so the arm measures what the donor's memory alone is worth to the receiver. The programs themselves transfer without loss in every direction: the receiver's first pass solves exactly the tasks the donor banked (118, 81, 121 and 81 tasks on AppWorld), and its Avg@4 goes above the donor's own Pass@4 because a transferred task is solved in every pass rather than discovered over

passes; continuing from the donor's frontier adds two to five tasks. A weaker donor's memory does not replace a stronger receiver's own search (GPT-4o reaches 86 tasks from Qwen3-8B's memory against 121 alone), which is why the arm is reported next to the receiver-alone column. Recovery-point transfer is lossy: a receiver continuing from the donor's recovery points while running its own protocol gains 3 to 9 tasks over running alone, more when the donor is the stronger model (7 and 9 tasks against 4 and 3 on AppWorld, 5 and 6 against 3 and 3 on BFCL-V3), because a stronger donor's frontier holds verified states of tasks the receiver does not reach alone.

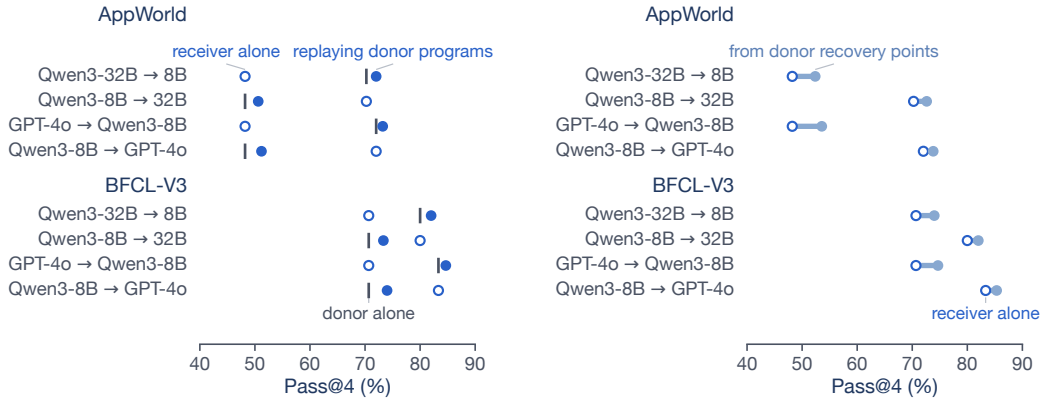


Figure A10: Cross-model transfer of experience (Pass@4). (a) Programs: a receiver that replays the donor's verified programs reaches the donor's Pass@4 (tick) without calling a model, plus the few tasks its own search adds, in all four directions including closed → open (GPT-4o → Qwen3-8B). (b) Recovery points: a receiver that continues from the donor's recovery points gains 3–9 tasks over running alone.

Table A20: Cross-model transfer in four directions (Pass@4 of the donor; Avg@4 and Pass@4 of the receiver alone, replaying the donor's programs, and continuing from the donor's recovery points). In the donor-programs arm the receiver makes no fresh attempts: it replays the donor's bank and continues only from the donor's frontier; "transf. / own" is the number of tasks solved by the replayed programs and by those continuations. In the donor-recovery-points arm the receiver runs its own protocol with the donor's recovery points added to its frontier; "+tasks" is its gain over running alone.

		Donor	Receiver alone		Donor programs			Donor recovery points		
Benchmark	Direction	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4	transf. / own	Avg@4	Pass@4	+tasks
AppWorld	Qwen3-32B → Qwen3-8B	70.24	34.67	48.21	71.13	72.02	118 / 3	36.61	52.38	7
AppWorld	Qwen3-8B → Qwen3-32B	48.21	54.91	70.24	49.55	50.6	81 / 4	55.95	72.62	4
AppWorld	GPT-4o → Qwen3-8B	72.02	34.67	48.21	72.62	73.21	121 / 2	37.35	53.57	9
AppWorld	Qwen3-8B → GPT-4o	48.21	61.16	72.02	49.85	51.19	81 / 5	62.05	73.81	3
BFCL-V3	Qwen3-32B → Qwen3-8B	80.0	60.33	70.67	81.0	82.0	120 / 3	61.83	74.0	5
BFCL-V3	Qwen3-8B → Qwen3-32B	70.67	71.5	80.0	72.17	73.33	106 / 4	72.5	82.0	3
BFCL-V3	GPT-4o → Qwen3-8B	83.33	60.33	70.67	84.0	84.67	125 / 2	62.33	74.67	6
BFCL-V3	Qwen3-8B → GPT-4o	70.67	75.33	83.33	72.5	74.0	106 / 5	76.33	85.33	3

H ABLATIONS

All ablations use Qwen3-8B; one component changes per row and everything else is the configuration of Table A1. The full model and the four search components of Table 2 are three-run means; the per-pass tables below are the reference run.

Component ablation, per pass. Table A21 is the per-pass version of Table 2. Removing verified replay leaves Pass@4 unchanged and costs 15.27 and 18.44 Avg@4 points against the three-run mean of the full model; that separation is the construction that keeps replay out of Pass@4. The text-only variant is the one row whose two metrics move in opposite directions against the main table: it still finds more tasks than No Memory, so its Pass@4 is above No Memory, but it holds so few of them that its Avg@4 gains almost nothing from having found them. Its solved set is easier than the full method's, which is why its per-task re-solve rate is higher than the summary arm of Figure 4a, which is measured on the full method's solved set. Pruning does not change accuracy on BFCL-V3 and costs one task on AppWorld, where a raw 14-action program exceeds the 30-step budget in pass 3.

Table A21: Component ablation with per-pass task counts (Qwen3-8B, reference run).

(a) AppWorld (168 tasks)

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
TaP (full)	25	54	73	81	34.67	48.21 (81/168)
w/o Verified State Localization (geometric depths)	25	50	66	76	32.29	45.24 (76/168)
w/o probes (prior from partial score only)	25	52	70	78	33.48	46.43 (78/168)
w/o marginal allocation (UCB over choices)	25	53	71	78	33.78	46.43 (78/168)
w/o context restore (environment state only)	25	52	70	79	33.63	47.02 (79/168)
w/o pruning (raw successful program replayed)	25	54	72	81	34.52	48.21 (81/168)
w/o verified replay (solved tasks regenerated)	25	37	36	31	19.20	48.21 (81/168)
skill-library reuse (programs as callable functions)	25	40	43	40	22.02	48.21 (81/168)
text-only experience (attempt stored as a summary)	25	30	28	26	16.22	38.10 (64/168)
retention only (no recovery search)	25	42	51	55	25.74	32.74 (55/168)

(b) BFCL-V3 (150 tasks)

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
TaP (full)	58	96	102	106	60.33	70.67 (106/150)
w/o Verified State Localization (geometric depths)	58	88	95	99	56.67	66.00 (99/150)
w/o probes (prior from partial score only)	58	92	100	104	59.00	69.33 (104/150)
w/o marginal allocation (UCB over choices)	58	93	101	104	59.33	69.33 (104/150)
w/o context restore (environment state only)	58	93	101	105	59.50	70.00 (105/150)
w/o pruning (raw successful program replayed)	58	96	102	106	60.33	70.67 (106/150)
w/o verified replay (solved tasks regenerated)	58	71	60	61	41.67	70.67 (106/150)
skill-library reuse (programs as callable functions)	58	84	82	85	51.50	70.67 (106/150)
text-only experience (attempt stored as a summary)	58	63	63	65	41.50	66.00 (99/150)
retention only (no recovery search)	58	74	83	89	50.67	59.33 (89/150)

Recovery-point rule. Table A22 compares the rule of Eq. 2 with five alternatives; branch success is the fraction of full attempts that solve their task. Restarting from the last step before the failure is the worst rule on both benchmarks: it preserves the most work and reproduces the failure, which is the deep end of the trade-off of Figure 5a. Taking the last depth of each plateau instead of the first costs one to two tasks: the same verified work is preserved but the remaining budget is shorter and the conditioning on the old path is stronger. On BFCL-V3 the first-failed-boundary rule alone recovers most of the gain, because a turn boundary is where a progress plateau starts.

Table A22: Recovery-point rule (Qwen3-8B, reference run). Only the recovery-point set changes; probing, allocation and replay are unchanged.

(a) AppWorld

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4	branch success %
first depth of each progress plateau (TaP)	25	54	73	81	34.67	48.21 (81/168)	3.50
geometric depths {L/3, L/2, 2L/3}	25	50	66	76	32.29	45.24 (76/168)	2.52
first failed boundary only	25	53	70	78	33.63	46.43 (78/168)	2.94
last step before failure ($d = L - 1$)	25	44	56	68	28.72	40.48 (68/168)	1.16
uniformly random depth (3 per trajectory)	25	48	63	73	31.10	43.45 (73/168)	2.03
last depth of each plateau	25	53	72	79	34.08	47.02 (79/168)	3.19

(b) BFCL-V3

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4	branch success %
first depth of each progress plateau (TaP)	58	96	102	106	60.33	70.67 (106/150)	6.47
geometric depths {L/3, L/2, 2L/3}	58	88	95	99	56.67	66.00 (99/150)	4.66
first failed boundary only	58	93	101	104	59.33	69.33 (104/150)	5.43
last step before failure ($d = L - 1$)	58	80	88	96	53.67	64.00 (96/150)	2.14
uniformly random depth (3 per trajectory)	58	86	93	100	56.17	66.67 (100/150)	3.75
last depth of each plateau	58	95	101	105	59.83	70.00 (105/150)	5.89

Allocation rule. Table A23 replaces the marginal-value rule of Eq. 5 by UCB, uniform-per-task and random allocation, and removes each of its terms in turn. The unsolved-probability term ϕ_i matters more than the repeat discount ρ^{n_u}: without it the pool concentrates on the few tasks whose best choice happens to draw a high sampled value. Removing prior propagation leaves Pass@4 unchanged on BFCL-V3 and costs one task on AppWorld, which is consistent with its role of saving probes on shallower points rather than changing which tasks are solved; the deletion rule behaves the same way, and without it the budget keeps returning to points that have already produced four deviating attempts with no progress.

Table A23: Allocation rule (Qwen3-8B, reference run). Only the allocation rule changes.

(a) AppWorld

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
marginal-value allocation (Eq. 5, TaP)	25	54	73	81	34.67	48.21 (81/168)
UCB over choices	25	53	71	78	33.78	46.43 (78/168)
uniform per task	25	52	69	77	33.18	45.83 (77/168)
random choice	25	50	66	75	32.14	44.64 (75/168)
no repeat discount (ρ^{n_u} removed)	25	54	72	80	34.38	47.62 (80/168)
no unsolved-probability term (ϕ_i removed)	25	53	70	78	33.63	46.43 (78/168)
no prior propagation between recovery points	25	54	72	80	34.38	47.62 (80/168)
no deletion rule (dead points keep their share)	25	53	72	80	34.23	47.62 (80/168)

(b) BFCL-V3

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
marginal-value allocation (Eq. 5, TaP)	58	96	102	106	60.33	70.67 (106/150)
UCB over choices	58	93	101	104	59.33	69.33 (104/150)
uniform per task	58	92	100	103	58.83	68.67 (103/150)
random choice	58	90	97	101	57.67	67.33 (101/150)
no repeat discount (ρ^{n_u} removed)	58	95	102	105	60.00	70.00 (105/150)
no unsolved-probability term (ϕ_i removed)	58	93	101	104	59.33	69.33 (104/150)
no prior propagation between recovery points	58	96	102	106	60.33	70.67 (106/150)
no deletion rule (dead points keep their share)	58	95	102	105	60.00	70.00 (105/150)

Probes and allocation under a smaller budget. Table A24 repeats the two allocation rows of Table 2 at 3, 6, 9 and 12 full attempts per unsolved task per round. The cost of removing the probes shrinks from 3.12 to 0.59 Avg@4 points on AppWorld and from 2.84 to 0.50 on BFCL-V3 as the budget grows, and the cost of the UCB rule from 2.23 to 0.44 and from 2.34 to 0.33: with twelve

attempts per task almost every recovery point is tried anyway, so the allocation has little left to decide.

Table A24: Probes and the marginal-value rule along the attempt budget (Qwen3-8B, reference run). The budget-9 rows are the rows of Table 2; the full-method rows are those of Table A34.

(a) AppWorld

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
budget 3, TaP (full)	25	45	60	70	29.76	41.67 (70/168)
budget 3, w/o probes	25	41	53	64	26.64	38.10 (64/168)
budget 3, w/o marginal allocation (UCB)	25	42	55	66	27.53	39.29 (66/168)
budget 6, TaP (full)	25	51	68	77	32.89	45.83 (77/168)
budget 6, w/o probes	25	48	63	73	30.95	43.45 (73/168)
budget 6, w/o marginal allocation (UCB)	25	49	64	74	31.40	44.05 (74/168)
budget 9, TaP (full)	25	54	73	81	34.67	48.21 (81/168)
budget 9, w/o probes	25	52	70	78	33.48	46.43 (78/168)
budget 9, w/o marginal allocation (UCB)	25	53	71	78	33.78	46.43 (78/168)
budget 12, TaP (full)	25	56	76	84	35.86	50.00 (84/168)
budget 12, w/o probes	25	55	74	83	35.27	49.40 (83/168)
budget 12, w/o marginal allocation (UCB)	25	55	75	83	35.42	49.40 (83/168)

(b) BFCL-V3

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
budget 3, TaP (full)	58	86	94	99	56.17	66.00 (99/150)
budget 3, w/o probes	58	81	87	94	53.33	62.67 (94/150)
budget 3, w/o marginal allocation (UCB)	58	82	88	95	53.83	63.33 (95/150)
budget 6, TaP (full)	58	92	100	104	59.00	69.33 (104/150)
budget 6, w/o probes	58	88	97	101	57.33	67.33 (101/150)
budget 6, w/o marginal allocation (UCB)	58	89	98	101	57.67	67.33 (101/150)
budget 9, TaP (full)	58	96	102	106	60.33	70.67 (106/150)
budget 9, w/o probes	58	93	101	104	59.00	69.33 (104/150)
budget 9, w/o marginal allocation (UCB)	58	94	101	103	59.33	69.33 (104/150)
budget 12, TaP (full)	58	98	105	108	61.50	72.00 (108/150)
budget 12, w/o probes	58	97	104	107	61.00	71.33 (107/150)
budget 12, w/o marginal allocation (UCB)	58	97	105	107	61.17	71.33 (107/150)

The same per-check feedback for a text memory. TaP reads the checker on intermediate states; the baselines of Table 1 see only the outcome of an attempt. Table A25 gives two text memories the same signal: Reflexion writes the failed checks of the attempt into its reflection, and Memp stores the passed and failed checks with every entry. The feedback is worth 1.6–1.9 Avg@4 and 1.1–2.0 Pass@4 points, about a tenth of the distance to TaP on Avg@4: the regenerated actions still differ from the ones that earned the checks, so the completed part of the attempt is still lost.

Table A25: Text memories given the per-check feedback of the evaluator (Qwen3-8B, means and standard deviations of three runs).

	AppWorld		BFCL-V3	
Method	Avg@4	Pass@4	Avg@4	Pass@4
Reflexion	17.76±0.60	35.52±0.91	42.33±0.87	63.11±0.77
Reflexion + per-check feedback	19.64±0.65	37.50±1.19	44.22±0.69	64.67±1.15
Memp	18.80±0.62	36.71±1.72	43.67±0.60	65.56±0.77
Memp + per-check feedback	20.39±0.68	38.10±1.19	45.28±0.63	66.67±0.67
TaP	**34.47**±0.62	**48.02**±1.50	**60.11**±0.54	**70.44**±1.02

Treatment of solved tasks. Table A26 changes only what happens to a solved task; the search is unchanged, so the set of tasks ever solved and hence Pass@4 is identical in every row. Every regenerating variant loses 13 to 23 Avg@4 points to regression, in the ordering of Figure 4a on the

same task set: the verbatim transcript is better than nothing and the summary is worse than nothing. Pruning removes 41.5% of the actions on AppWorld (11.8 to 6.9) and 24.5% on BFCL-V3 (9.4 to 7.1), where turn structure leaves less to remove.

Table A26: Treatment of solved tasks (Qwen3-8B, reference run). Regression is the definition of Figure 4b on this task set; program length is the mean number of actions in the stored artefact; calls are model calls per solved task per later pass.

(a) AppWorld

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4	regression %	program length	calls
verified pruned program (TaP)	25	54	73	81	34.67	48.21 (81/168)	0.0	6.9	0
raw successful program (no pruning)	25	54	72	81	34.52	48.21 (81/168)	0.0	11.8	0
verbatim transcript in context, regenerate	25	38	38	33	19.94	48.21 (81/168)	65.12	—	18.1
no experience in context, regenerate	25	37	36	31	19.20	48.21 (81/168)	68.83	—	18.1
summary of the attempt in context, regenerate	25	35	33	26	17.71	48.21 (81/168)	74.69	—	18.1

(b) BFCL-V3

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4	regression %	program length	calls
verified pruned program (TaP)	58	96	102	106	60.33	70.67 (106/150)	0.0	7.1	0
raw successful program (no pruning)	58	96	102	106	60.33	70.67 (106/150)	0.0	9.4	0
verbatim transcript in context, regenerate	58	75	67	68	44.67	70.67 (106/150)	36.79	—	11.6
no experience in context, regenerate	58	71	60	61	41.67	70.67 (106/150)	43.87	—	11.6
summary of the attempt in context, regenerate	58	65	51	52	37.67	70.67 (106/150)	52.83	—	11.6

Temperature set. Table A27 varies the temperature set of probes and full attempts at a constant budget. The default set is better than any single temperature or shifted set by one to four tasks, and Table A28 shows why it is kept rather than collapsed: successful full attempts are spread over the three arms (29.7–38.1% each) and no single temperature dominates, while the deviation rate that the allocation estimates per temperature differs across arms (Figure A4).

Table A27: Temperature set (Qwen3-8B, reference run). The default set differs by benchmark.

(a) AppWorld

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
{0.9} only	25	52	69	77	33.18	45.83 (77/168)
{0.7, 0.9, 1.1} (default)	25	54	73	81	34.67	48.21 (81/168)
{1.1, 1.3}	25	52	70	78	33.48	46.43 (78/168)
{0.5, 0.7, 0.9}	25	53	70	78	33.63	46.43 (78/168)

(b) BFCL-V3

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
{1.1} only	58	92	99	102	58.50	68.00 (102/150)
{0.9, 1.1, 1.3} (default)	58	96	102	106	60.33	70.67 (106/150)
{0.7, 0.9, 1.1}	58	93	101	104	59.33	69.33 (104/150)
{1.3, 1.5}	58	91	98	103	58.33	68.67 (103/150)

Table A28: Successful full attempts by temperature arm under the default set (Qwen3-8B). Shares are over successful full attempts, not over tasks, since several attempts can solve the same task.

Variant	arm 1	arm 2	arm 3	total successful full attempts
AppWorld {0.7, 0.9, 1.1}	38 (34.2%)	40 (36.0%)	33 (29.7%)	111
BFCL-V3 {0.9, 1.1, 1.3}	35 (31.0%)	43 (38.1%)	35 (31.0%)	113

Frontier retention rule. Table A29 compares the rule of Section 3.2 (highest partial score, longer trajectory as tie-break) with three alternatives at frontier size 4. The tie-break by length is worth about one task on AppWorld and nothing on BFCL-V3, where trajectories of equal partial score usually have equal length because progress is quantised by turns.

Table A29: Frontier retention rule at size 4 (Qwen3-8B, reference run).

Variant	AppWorld Avg@4	AppWorld Pass@4	BFCL-V3 Avg@4	BFCL-V3 Pass@4
highest partial score, longer trajectory as tie-break (TaP)	34.67	48.21 (81/168)	60.33	70.67 (106/150)
highest partial score, shorter trajectory as tie-break	34.23	47.62 (80/168)	60.17	70.67 (106/150)
most recent trajectories	33.78	47.02 (79/168)	59.83	70.00 (105/150)
random trajectories	33.48	46.43 (78/168)	59.50	69.33 (104/150)

I HYPER-PARAMETER SWEEPS

Figure A11 sweeps one hyper-parameter at a time with everything else at the default; Tables A30–A35 give Pass@4 and the interaction counts. Every sweep plateaus at or near the default except the full-attempt budget, which is monotone with a diminishing slope. One probe per choice leaves the estimate too noisy on the deepest points, where the deviation rate is lowest; beyond two probes the set of solved tasks stops changing while probe episodes grow linearly. Two probe actions rarely reach a new check, so improvement is under-detected, and eight turn a probe into half of a full attempt without changing which tasks are solved. A weak prior lets a single lucky full attempt dominate a choice and a strong one slows the update from full-attempt outcomes. With one recovery point per task the method degenerates into a fixed rule, and eight points reach the same Pass@4 as four because most frontier trajectories have fewer than four plateaus (Table A10). One frontier trajectory misses the recovery points that only another attempt reached, and beyond four the added trajectories are mostly duplicate prefixes that only add offline evaluation cost. The attempt budget is monotone: twelve full attempts per unsolved task per round give 35.86 and 61.50 Avg@4 against 34.67 and 60.33 at the default of nine, at a proportionally larger cost (Table A34); the default is the budget used everywhere else in the paper and Table A2 reports what it costs.

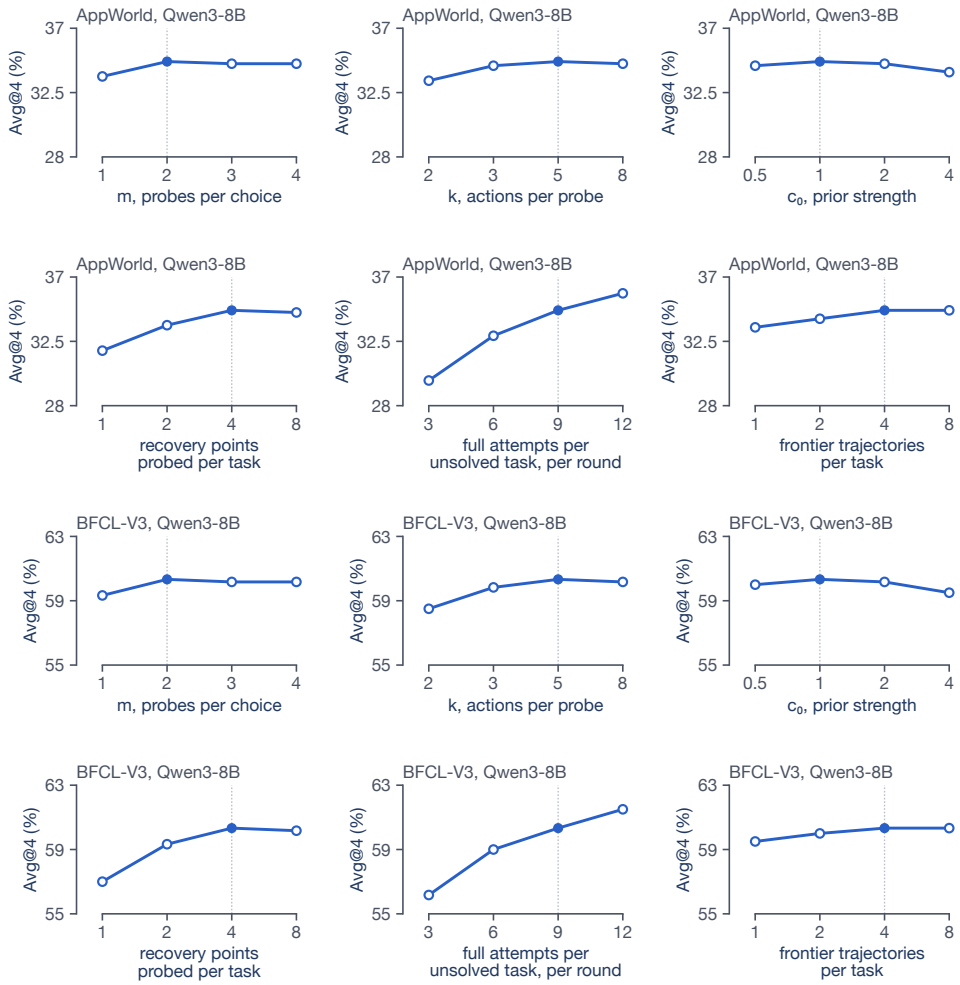


Figure A11: Hyper-parameter sweeps, Avg@4, Qwen3-8B, AppWorld (top two rows) and BFCL-V3 (bottom two rows); the filled marker and dotted line mark the default used everywhere else in the paper. All sweeps plateau at or near the default except the full-attempt budget, which is monotone with a diminishing slope.

Table A30: Probes per choice m (Qwen3-8B, reference run; the default row is the reference run of Table 1.)

(a) AppWorld

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4	probe episodes
$m = 1$	25	52	70	79	33.63	47.02 (79/168)	1006
$m = 2$ (default)	25	54	73	81	34.67	48.21 (81/168)	1991
$m = 3$	25	54	72	81	34.52	48.21 (81/168)	2993
$m = 4$	25	53	73	81	34.52	48.21 (81/168)	3992

(b) BFCL-V3

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4	probe episodes
$m = 1$	58	93	101	104	59.33	69.33 (104/150)	582
$m = 2$ (default)	58	96	102	106	60.33	70.67 (106/150)	1147
$m = 3$	58	96	101	106	60.17	70.67 (106/150)	1727
$m = 4$	58	95	102	106	60.17	70.67 (106/150)	2304

Table A31: Probe actions k (Qwen3-8B, reference run; the default row is the reference run of Table 1.)

(a) AppWorld

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
$k = 2$	25	52	69	78	33.33	46.43 (78/168)
$k = 3$	25	54	72	80	34.38	47.62 (80/168)
$k = 5$ (default)	25	54	73	81	34.67	48.21 (81/168)
$k = 8$	25	53	73	81	34.52	48.21 (81/168)

(b) BFCL-V3

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
$k = 2$	58	91	99	103	58.50	68.67 (103/150)
$k = 3$	58	94	102	105	59.83	70.00 (105/150)
$k = 5$ (default)	58	96	102	106	60.33	70.67 (106/150)
$k = 8$	58	95	102	106	60.17	70.67 (106/150)

Table A32: Prior strength c_0 (Qwen3-8B, reference run; the default row is the reference run of Table 1.)

(a) AppWorld

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
$c_0 = 0.5$	25	54	72	80	34.38	47.62 (80/168)
$c_0 = 1$ (default)	25	54	73	81	34.67	48.21 (81/168)
$c_0 = 2$	25	53	73	81	34.52	48.21 (81/168)
$c_0 = 4$	25	53	71	79	33.93	47.02 (79/168)

(b) BFCL-V3

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
$c_0 = 0.5$	58	95	102	105	60.00	70.00 (105/150)
$c_0 = 1$ (default)	58	96	102	106	60.33	70.67 (106/150)
$c_0 = 2$	58	95	102	106	60.17	70.67 (106/150)
$c_0 = 4$	58	94	101	104	59.50	69.33 (104/150)

Table A33: Recovery points probed per task (Qwen3-8B, reference run; the default row is the reference run of Table 1.)

(a) AppWorld

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
recovery points per task = 1	25	49	65	75	31.85	44.64 (75/168)
recovery points per task = 2	25	52	70	79	33.63	47.02 (79/168)
recovery points per task = 4 (default)	25	54	73	81	34.67	48.21 (81/168)
recovery points per task = 8	25	54	72	81	34.52	48.21 (81/168)

(b) BFCL-V3

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
recovery points per task = 1	58	88	96	100	57.00	66.67 (100/150)
recovery points per task = 2	58	93	101	104	59.33	69.33 (104/150)
recovery points per task = 4 (default)	58	96	102	106	60.33	70.67 (106/150)
recovery points per task = 8	58	95	102	106	60.17	70.67 (106/150)

Table A34: Full attempts per unsolved task per round (Qwen3-8B, reference run; the default row is the reference run of Table 1.)

(a) AppWorld

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4	full attempts total
full attempts per unsolved task per round = 3	25	45	60	70	29.76	41.67 (70/168)	1122
full attempts per unsolved task per round = 6	25	51	68	77	32.89	45.83 (77/168)	2160
full attempts per unsolved task per round = 9 (default)	25	54	73	81	34.67	48.21 (81/168)	3168
full attempts per unsolved task per round = 12	25	56	76	84	35.86	50.00 (84/168)	4164

(b) BFCL-V3

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4	full attempts total
full attempts per unsolved task per round = 3	58	86	94	99	56.17	66.00 (99/150)	636
full attempts per unsolved task per round = 6	58	92	100	104	59.00	69.33 (104/150)	1200
full attempts per unsolved task per round = 9 (default)	58	96	102	106	60.33	70.67 (106/150)	1746
full attempts per unsolved task per round = 12	58	98	105	108	61.50	72.00 (108/150)	2268

Table A35: Frontier trajectories per task (Qwen3-8B, reference run; the default row is the reference run of Table 1.)

(a) AppWorld

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
frontier trajectories = 1	25	52	70	78	33.48	46.43 (78/168)
frontier trajectories = 2	25	53	71	80	34.08	47.62 (80/168)
frontier trajectories = 4 (default)	25	54	73	81	34.67	48.21 (81/168)
frontier trajectories = 8	25	54	73	81	34.67	48.21 (81/168)

(b) BFCL-V3

Variant	pass 1	pass 2	pass 3	pass 4	Avg@4	Pass@4
frontier trajectories = 1	58	94	101	104	59.50	69.33 (104/150)
frontier trajectories = 2	58	95	102	105	60.00	70.00 (105/150)
frontier trajectories = 4 (default)	58	96	102	106	60.33	70.67 (106/150)
frontier trajectories = 8	58	96	102	106	60.33	70.67 (106/150)

J BFCL-V3 SUBCATEGORIES AND SCALE

Other BFCL-V3 multi-turn subcategories. Table A36 evaluates Qwen3-8B on the three other multi-turn subcategories with the published subcategory protocol: the base subcategory uses 50 tasks for the initial pool and 150 for evaluation, the other three are evaluated on all 200 tasks each, and the No Memory and ReMe (fixed) rows are the published numbers. Miss Func is the hardest subcategory for every method, because the agent has to notice that no available function can serve the request; that judgement happens in the first turn and there is no verified state to recover to, so the gain there is 8.00 points against 13.50 on Miss Param and 12.50 on Long Context, where the failure usually occurs after some turns have already passed and replay skips them.

Table A36: BFCL-V3 multi-turn subcategories, Pass@4 (Qwen3-8B). TaP is a single run.

Method	Overall (750)	Base (150)	Miss Param (200)	Miss Func (200)	Long Context (200)
No Memory	37.78	59.55	31.00	19.00	47.00
ReMe (fixed)	43.02	65.77	38.50	21.50	52.00
TaP	49.07	70.67 (106/150)	44.50 (89/200)	27.00 (54/200)	59.50 (119/200)

Branch success across backbones. Table A37 reports two rates from the accounting of Table A2: the fraction of full attempts that solve their task, and the fraction that are the first to solve a task. Both are low in absolute terms, because a full attempt has to finish the whole remainder of the task and most of the pool it is spent on cannot be finished at all. The branch success rate peaks at Qwen3-14B on both benchmarks (3.50 to 6.75 then 6.15 on AppWorld; 6.47 to 11.40 then 10.56 on BFCL-V3), because the tasks still unsolved by the largest backbone are dominated by tasks that never reach a verified state; Mistral-Small-3.2-24B sits between the 8B and 14B rates, the position it occupies in Table 1. Several full attempts can solve the same task in the same round, so the branch success rate exceeds the first-solve rate; the number of successful attempts per newly solved task is 2.12 to 4.06.

Table A37: Full attempts and their conversion on every setting (reference run): successful full attempts, the branch success rate, tasks first solved by a full attempt, the first-solve rate, and successful attempts per newly solved task.

Benchmark	Backbone	full attempts	successful	branch succ. (%)	first solves	first-solve rate (%)	succ. per new task
AppWorld	Qwen3-8B	3168	111	3.5	40	1.26	2.77
AppWorld	Qwen3-14B	2385	161	6.75	63	2.64	2.56
AppWorld	Qwen3-32B	2277	140	6.15	66	2.9	2.12
AppWorld	Mistral-Small-3.2-24B	2907	131	4.51	50	1.72	2.62
BFCL-V3	Qwen3-8B	1746	113	6.47	45	2.58	2.51
BFCL-V3	Qwen3-14B	1368	156	11.4	54	3.95	2.89
BFCL-V3	Qwen3-32B	1269	134	10.56	33	2.6	4.06
BFCL-V3	Mistral-Small-3.2-24B	1647	125	7.59	43	2.61	2.91

K QUALITATIVE EXAMPLES

All prompts are AppWorld `test_normal` or BFCL-V3 `multi_turn_base` evaluation tasks, and the memory pools of the baselines are built on the training split. Each example shows the same task under No Memory (a fresh attempt), Reflexion (a fresh attempt carrying the reflection written after its previous failure) and TaP (an attempt from a recovery point, or a replay). Actions are the code blocks the agent emitted; observations are the environment's returns, truncated with "…". The five examples cover recovery from a verified mid-trajectory state, the probes rejecting a deeper recovery point, the same experience as text and as a program, recovery at a turn boundary, and verified pruning.

Example 1 (recovery from a verified mid-trajectory state (AppWorld, Qwen3-8B)). **Prompt**: Add all the songs in my Spotify liked songs that are by an artist I follow to the playback queue.

No Memory	Steps 1-6 log in and fetch `liked_songs` and `following` (progress 3/5 checks). Step 7: `for song in liked_songs: if song["artist_id"] in following: ...` — `following` is a list of dicts, so the test is always false and the queue stays empty. Steps 8-10 re-fetch the same lists and call `complete_task()`. Final 3/5, failed.
Reflexion	Reflection after the failure: "I compared an id against a list of dictionaries. Build a set of followed artist ids first." The next attempt starts over: steps 1-4 differ from the recorded ones (it opens Spotify before reading the passwords, hits an auth error, retries), the id set is built correctly at step 8, and the attempt then exhausts the 30-step budget re-paginating `show_liked_songs`. Final 2/5, failed.
TaP	Recovery point at depth 5, the first depth reaching the 3-check plateau. Replay of the recorded steps 1-5 restores the state with both lists fetched. A probe from that state at T=1.1 leaves the recorded path at its first action. The full attempt generates only the suffix: `followed = {a["artist_id"] for a in following}` then `for s in liked_songs: if s["artist_id"] in followed: apis.spotify.add_to_queue(access_token=token, song_id=s["song_id"])`, then `complete_task()`. Final 5/5, solved with 9 actions; the pruned program has 5 actions and replays in 7 seconds.

Why this example. the failed attempt had already done the expensive part correctly, two paginated fetches behind a login. TaP keeps it; the regenerated attempt loses it.

Example 2 (probes reject the deeper recovery point (AppWorld, Qwen3-8B)). **Prompt**: Mark every task in my Todoist project "Move" that is already done in my SimpleNote checklist as completed.

No Memory	Steps 1-7 log in to both apps, read the note and list the project's tasks (progress 2/4). Step 8 marks the first matching task complete. Step 9 matches by task title against the note's raw text and closes a task that is not in the checklist; steps 10-13 try the same title match twice more and complete the episode. Final 2/4, failed.
Reflexion	Reflection: "Match on the checklist items, not on the raw note text." The next attempt marks the first task complete, then calls `update_task(task_id=..., completed=True)` with the note id instead of the task id, gets an error, and completes. Final 2/4, failed.
TaP	Two recovery points: depth 7 (both apps read, 2/4 verified) and depth 8 (first task closed, 3/4 verified). Probes: from depth 8, three of four continuations reproduce the recorded title-matching call and its observation, and the one that deviates makes no new progress, so $\rho = 0.25$ and $g = 0.00$ for that point; from depth 7, three of four deviate and one reaches 4/4 within five actions. The full attempts go to depth 7: the continuation parses the checklist into items first, then closes both matching tasks by id. Final 4/4, solved.

Why this example. the deeper state preserves more verified work, but the model re-issues the action that failed from it. The probe measures exactly that and the budget moves to the shallower point, which is the trade-off of Method §3.3.

Example 3 (the same experience as text and as a program (AppWorld, Qwen3-32B)). **Prompt**: Create a note titled "Groceries" in SimpleNote with the items from my last Amazon order, one per line.

No Memory	Solved in pass 2 with 8 actions. In pass 3 the fresh attempt sorts the items alphabetically and appends a trailing newline; the note-content check fails. Final 3/4 checks, a solved task lost.
Reflexion	The pass-2 success is summarised as "Fetch the last order, then create the note with one item per line." Pass 3 regenerates from that summary: `content = "\n".join(sorted(items))`. Final 3/4, lost again.
TaP	The pass-2 program, pruned from 8 actions to 5 (login, `show_orders(...)[0]`, `show_order(order_id=...)`, `create_note(title="Groceries", content="\n".join(i["name"] for i in items))`, `complete_task()`), is replayed in passes 3 and 4 with no model call. Final 4/4 in both.

Why this example. the summary is accurate and the regenerated attempt still differs in the detail the check cares about. A program cannot differ.

Example 4 (turn-boundary recovery (BFCL-V3 multi-turn, Qwen3-8B))**. Prompt**(three user turns): (1) "Go to the `reports` directory and show me what is in it." (2) "Create `summary.txt` containing the word counts of every `.md` file there." (3) "Move `summary.txt` into `archive/` and confirm it is there."

No Memory	Turn 1 passes (`cd(folder="reports"), ls()`). Turn 2 calls `wc(file_name=...)` for each file but writes the result with `echo(...)` into the parent directory. Turn 3 then moves a file that does not exist in `reports`. Turn vector [1, 0, 0].
Reflexion	Reflection: "Write summary.txt inside the reports directory." The next attempt passes turn 1, writes the file in the right place but counts lines rather than words (`mode="l"`), and fails the content check. Turn vector [1, 0, 0].
TaP	The progress profile [1, 0, 0] puts the recovery point at the end of turn 1. Replay restores the shell inside `reports` with the same listing, and the attempt regenerates turn 2 only: `wc(file_name=f, mode="w")` per file, then `echo(...)` into `reports/summary.txt`; turn 3 then succeeds. Turn vector [1, 1, 1], solved, and replayed in later passes.

Why this example. the per-turn checker makes the plateau explicit, recovery starts exactly at the first failed turn, and the two earlier turns are never regenerated.

Example 5 (pruning keeps only the verified actions (AppWorld, Mistral-Small-3.2-24B))**. Prompt**: Like every song in my "Road Trip" playlist on Spotify.

No Memory	11 actions: two exploratory `show_playlists` calls with different page sizes, a `show_playlist` on the wrong playlist, then the right one, then a like loop. Solved in pass 1; in pass 2 the fresh attempt likes the songs of a playlist with a similar name and fails.
Reflexion	No reflection is written after a success, so pass 2 regenerates from scratch and repeats the No Memory failure.
TaP	The 11-action success is reduced by verified deletion to 5 actions (login, `show_playlists()`, the correct `show_playlist(playlist_id=...)`, the like loop, `complete_task()`); each deletion is accepted only if re-execution still passes all four checks. The 5-action program is replayed in passes 2 to 4.

Why this example. pruning removes the exploration that happened to precede the solution, and verification guarantees that what remains still solves the task.