CAVRA: CAUSAL ADAPTIVE VERIFIED REMASKING FOR SELF-EVOLVING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents are increasingly deployed as long-horizon controllers, yet they are almost always served with frozen weights, so any lasting improvement must live in the external textual policy rather than in the parameters. Turning a failed trajectory into a policy that transfers remains obstructed by two protocol gaps: self-evolution still repairs failure at episode granularity, rewriting the whole reply rather than the causal span, and the next policy is still authored from the transcript rather than induced. We therefore propose **CAVRA**, a remask and recover framework in which the same frozen model is both the student that acts and the teacher that localizes failure. A privileged remask spends a bounded mask budget on the high-credit control span, reading the world's own stall signature rather than a second judge. The student then inpaints only those spans and scores each fill with a progress-aligned probe confined to the error region, so the official first-pass verdict is never overwritten. Utility-gated distillation crystallizes a skill from remask kinds and failure reasons, or from already-corrected local fixes, after instance facts have been stripped, and family-conditioned routing injects only the matching skeleton. Extensive experiments, including a matched-backbone comparison against ten baselines, module ablations, and adaptation-cost measurements, show that CAVRA improves interactive control substantially, leading the strongest baseline by 40.6% relative success on ALFWorld. Our anonymized code is available at: `https://anonymous.4open.science/r/CAVRA-F1C8`.

1 INTRODUCTION

Large language model (LLM) agents are rapidly becoming the default interface for long-horizon work: they call tools and execute multi-step plans in interactive worlds (Yao et al., 2023b; Trivedi et al., 2024; Yao et al., 2025; Barres et al., 2025). In most deployments the backbone is frozen—a served LLM that is too expensive to retrain—while the surrounding textual harness can still be revised from execution experience (Hu et al., 2025; Yuksekgonul et al., 2025). This non-parametric route to self-evolution needs no access to weights, but converting a failed trajectory into a persistent policy that later tasks can actually use is substantially harder than writing another textual reflection.

Recent work along this route falls into three families. *Verbal critique and deliberative search* treats a failed attempt as material for another full generation: Tree of Thoughts explores intermediate thoughts as a search tree before committing to an answer (Yao et al., 2023a); CRITIC validates and revises an output through tool-interactive critiquing (Gou et al., 2024); and LATS unifies reasoning, acting, and planning by wrapping a language agent in Monte Carlo tree search with self-reflection (Zhou et al., 2024). *Experience libraries and reusable skills* instead keep what the agent did: Voyager grows an executable skill library from embodied programs (Wang et al., 2023); Agentic Plan Caching extracts structured plan templates at test time and reuses them on similar tasks (Zhang et al., 2025); and Nemori decides what deserves memory by distilling experiences that existing knowledge fails to predict (Ma et al., 2026). *Harness- and skill-level optimization* edits the scaffolding around a frozen model: SAGE trains a skill library with skill-augmented reinforcement learning and sequential rollouts (Wang et al., 2026b), while FlowEvo co-evolves inference-time workflows with a persistent bank of executable skills (Ren et al., 2026).

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

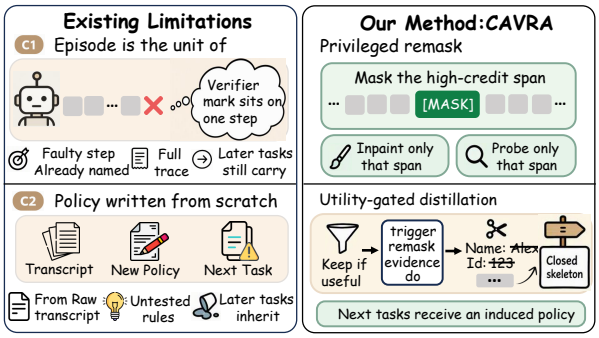


Figure 1: Limitations beside repairs in CAVRA.

Two protocol gaps remain unnamed. Figure 1 places each gap beside the repair in CAVRA. **(C1) Failure is still repaired at episode granularity.** Textual self-evolution after a failure still regenerates the whole episode (Nguyen et al., 2026; Liu et al., 2026), rewriting every step even when the native verifier has isolated a local defect, instead of marking the verbatim spans that caused that verifier to fail. The extra cost is another full reasoning trace rather than a local edit, and later tasks inherit an unrepaired control span. **(C2) The next policy is still authored from scratch.** After an attempt, self-evolution still authors a new critique, policy string, or skill from the raw transcript (Ray & Goyal, 2026; Shang et al., 2026), an unconstrained write that invents undemonstrated rules and contaminates later updates on the same stream. A skill written without an acceptance test on a demonstrated local repair is a new prompt, not an induced policy that later tasks can reuse.

To close these gaps we propose CAVRA, an on-policy textual self-distillation framework in which one frozen model is both the student that acts and the teacher that localizes failure. **Privileged remask** answers C1: the same model proposes the smallest verbatim spans that caused the native verifier to fail, then spends the mask budget on the high-credit control span read from the world's own stall signature rather than from a second judge. The official first-pass verdict is left untouched, so the remask locates a defect instead of rewriting the score. **Span-confined recover** then fills only the occluded spans, keeping unmasked text intact so the extra call does not regenerate the episode, and a bounded loop of progress-aligned probes scores the error region alone. If any probe accesses, the loop stops; if none does, the episode still ends after the recover budget is spent. **Utility-gated distillation** answers C2: it does not ask a learner to compose a new policy from the transcript. A skill is crystallized only when that recover was useful—loop access, or a real local fix—from remask kinds together with failure reasons or already-corrected fragments, after names and identifiers have been stripped. A family-conditioned router then emits a closed, instance-free skeleton and ranks the bounded store, so the next task is a single student generation under the frozen policy.

Overall, our contributions are summarized as follows:

- We propose CAVRA, an on-policy textual self-distillation framework in which one frozen model is both the student that acts and the teacher that localizes first-pass failure. It closes the two gaps by editing the already-emitted reply at span granularity and inducing the next transferable policy from remask evidence rather than authoring it from the raw transcript.
- We introduce privileged remask, which manufactures a span-level target on the student's own states, and utility-gated distillation, which writes a skill from remask kinds and failure evidence only when the recover accessed or made local progress, after instance facts have been stripped.
- Across question answering, mathematics, program synthesis, and interactive control, CAVRA improves held-out success against ten matched-backbone baselines, while scoring each eval task in one student pass; saturated-text accuracy is reported as a diagnostic and remains competitive. The largest lift is on interactive control: a 40.6% relative gain over the strongest ALFWorld baseline.

2 Methodology

This section presents the technical details of CAVRA. Figure 2 shows the overall framework.

2.1 Sealed Weights and On-Policy Supervision

Self-evolution is more reliable when the object that improves is not also the object that scores the improvement. This subsection seals that contract: the backbone stays frozen, only an external textual skill store may change, and the benchmark's own first-pass verdict is treated as an immutable instrument rather than as a quantity the update loop is allowed to edit.

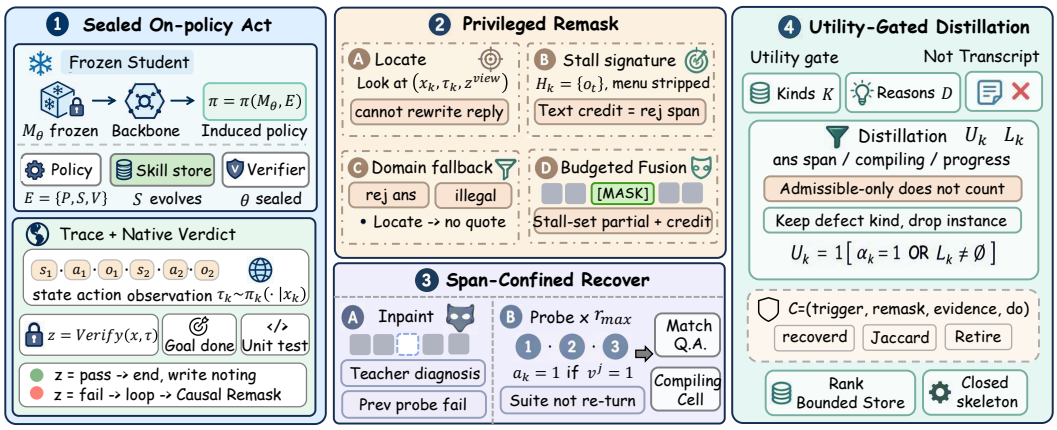


Figure 2: Architecture of the proposed CAVRA framework for self-evolving agents.

2.1.1 Frozen External Policy State

Let M_θ denote a frozen autoregressive backbone, with sealed weights θ that never receive a gradient during evolution. An agent policy state is an external, versioned triple $E_k = \{P_k, \mathcal{S}_k, \mathcal{V}_k\}$, where P_k is the system policy in force at step k, $\mathcal{S}_k$ is the bounded skill store that will later absorb distilled text, and $\mathcal{V}_k$ is the environment's own verifier. Only $\mathcal{S}_k$ evolves across tasks; θ is invariant. The induced student policy π_k and the sealed-weight constraint are:

$$\pi_k = \pi_{M_\theta, E_k}, \quad \theta_{k+1} = \theta_k = \theta, \tag{1}$$

so every later task is executed by the same model under a possibly richer store. The *teacher is the student*: no separate or privileged model is queried. Evolve tasks may update $\mathcal{S}_k$; tasks reserved for scoring are executed once under the frozen final store and never enter the remask loop.

2.1.2 On-Policy Trace and Native Verdict

For an incoming task x_k the student produces one on-policy trace from the input and the skills in force. The trace τ_k is the sequence of states s_t, actions a_t and observations o_t over a horizon T_k, and the native verdict z_k is whatever the deployment already has—exact match, unit tests, world success, or task-goal completion—together with only the telemetry a served agent would see:

$$\tau_k = \{(s_t, a_t, o_t)\}_{t=1}^{T_k} \sim \pi_k(\cdot \mid x_k), \quad z_k = \mathrm{Verify}(x_k, \tau_k). \tag{2}$$

This official score is never rewritten: the remask loop exists to locate a defect and to distill a skill, not to replace the first-pass metric. If z_k succeeds, the episode ends and no skill is written. Gold answers, hidden tests and LLM judges never enter the subsequent prompts.

2.2 Privileged Remask of the Failed Reply

A weight-space teacher is typically given a privileged view of the same state. Here the privilege is the native verifier together with the already-emitted reply: the teacher may carve a causal occlusion into that reply, but may not rewrite the official score or regenerate the episode. This subsection builds that privileged remask so the student later fills a hole rather than replaying a trace.

2.2.1 Privileged Span Proposal

On a first-pass failure the same model is re-queried as a teacher. It sees the verifier result and the full student context—task, steps, observations, and final output—and cannot rewrite the reply. It marks the smallest spans that caused z_k to fail. Writing z_k^{view} for the verifier telemetry visible to the teacher—the official outcome and the error region, never hidden tests or gold labels—the teacher proposal R_k^{loc} is a set of at most m_k=3 pairs (r_i, d_i), where r_i is a verbatim quote from the student's own output and d_i is a short reason that must not name the gold label:

$$R_k^{\mathrm{loc}} = \mathrm{Locate}\big(M_\theta;\, x_k, \tau_k, z_k^{\mathrm{view}}\big), \quad |R_k^{\mathrm{loc}}| \le m_k. \tag{3}$$

If the teacher returns no usable quote, a domain fallback remasks the native error region—the rejected answer, the submitted body, or the last illegal command—rather than the whole output. Remasking the entire reply is forbidden: the point is to avoid paying for another full reasoning trace.

2.2.2 STALL-SIGNATURE CREDIT ASSIGNMENT

Teacher quotes are a proposal, not a credit assignment. On interactive tasks almost all useful evolution budget sits in the *control* span rather than in a uniform rewrite of the whole trace; splitting that budget across low-credit spans freezes the update. Let $H_k = \{o_t\}$ be the world's own observation stream on τ_k, with the admissible-command menu stripped so that a changing menu cannot masquerade as progress. Read from H_k, a command a_t is *stalled* when o_t repeats an earlier reading or lies in $\mathcal{E} = \{\emptyset, \text{illegal}, \text{nothing}\}$. The target $a_k^\star$ is the mode of the last $w{=}4$ stalled commands:

$$\text{stall}(o_t) = \mathbf{1}\big[o_t \in \{o_i\}_{i<t} \cup \mathcal{E}\big], \quad a_k^\star = \text{mode}\big(\text{tail}_w\{a_t : \text{stall}(o_t)\}\big). \tag{4}$$

On text and code domains the same idea reduces to the verifier-rejected answer or body already required above: that span is the only region z_k actually scored, so it is the only region that can carry credit. The student remask target on interactive domains is therefore $a_k^\star$ itself, so recover still fills a single stalled command rather than regenerating the episode.

2.2.3 BUDGETED REMASK FUSION

Locate then unions this causal span with the teacher's quotes, the causal span taking the first mask-budget slot. The merge operator Merge prefers $a_k^\star$ and keeps teacher quotes only when they name a distinct, still-budgeted region; $d^\star$ is a short diagnosis attached to the stall span:

$$R_k = \text{Merge}\big(\{(a_k^\star, d^\star)\},\, R_k^{\text{loc}}\big), \quad \tilde{a}_k = \text{MASK}(a_k; R_k). \tag{5}$$

The resulting remask is what the next subsection fills and, if the fill is useful, what it distills into a skill. Interactive partials that never leave the stall set are not admitted as credit-bearing repairs.

2.3 TEXTUAL SELF-DISTILLATION UNDER THE EVOLUTION OBJECTIVE

A remask that is never filled teaches nothing, and a skill authored from the raw transcript is a new prompt, not a distilled policy. We now write the evolution objective that this last stage is meant to serve. Let μ be the measure over incoming tasks. Self-evolution maximises the expected native score of the skill store under the sealed backbone:

$$\mathcal{J}(\mathcal{S}) = \mathbb{E}_{x\sim\mu}\big[\text{Verify}(x, \tau),\ \tau \sim \pi_{M_\theta, E_\mathcal{S}}\big], \tag{6}$$

subject to the backbone θ remaining fixed and z_k remaining an immutable sample of $\mathcal{J}$ rather than a quantity the store update may edit. The remainder of this subsection then realises one constrained step on $\mathcal{J}$: a span-confined inner improvement, a utility gate that accepts a distilled skill, and a family-conditioned deployment of the current store.

2.3.1 SPAN-CONFINED INNER IMPROVEMENT

Unmasked text is held fixed, so the inner problem lives only on the remasked spans. Writing $\Omega(\tilde{a}_k)$ for the feasible fills—replies that agree with $\tilde{a}_k$ off the mask— and writing ξ_k for a residual that carries the teacher's diagnosis and, after the first round, the previous probe failure, the recover a_k' is a one-step improvement drawn from the same frozen model:

$$\Omega(\tilde{a}_k) = \big\{a :\ a \equiv \tilde{a}_k \text{ off } R_k\big\}, \quad a_k' = M_\theta\big(\tilde{a}_k,\ \xi_k\big) \in \Omega(\tilde{a}_k). \tag{7}$$

A loop of at most $k_{\max}{=}3$ rounds repeats this improvement against a *targeted* probe of the error region. The official suite is not re-run: z_k remains sealed. The probe $v_k^{(j)}$ is the environment's instrument on the remasked region—exact match on a recovered answer, implicated unit tests on remasked code, an unused interact verb outside the stall window, or a newly compiling cell—and the loop *accesses* when any round improves that local surrogate of $\mathcal{J}$:

$$v_k^{(j)} = \text{Probe}(\tilde{a}_k, a_k'^{(j)}), \quad A_k = \mathbf{1}\Big[\max_{j\le k_{\max}} v_k^{(j)} = 1\Big]. \tag{8}$$

On embodied control the surrogate is therefore *progress-aligned*: a merely legal fill—an unused navigation step, a repeated look, or anything already in the stall set—does not access. Only an admissible unused interact verb counts as a local improvement of the world, even when the outer sample z_k stays a failure. Writing $\mathcal{A}_t$ for the admissible command set at step t, that local check is itself a feasibility verdict rather than an instruction to the teacher:

$$\mathrm{Exec}(a') = \mathbf{1}\big[a' \in \mathcal{A}_t \ \wedge \ a' \text{ unused at } t\big]. \tag{9}$$

Navigation and look commands can be feasible while leaving the world reading unchanged; they are not credit-bearing repairs. If any inner round accesses, the loop stops; if none does, the episode ends when the inner budget is spent. In both cases z_k is left unchanged.

2.3.2 Utility-Gated Store Update

Let K_k be the remask *kinds* (answer, reasoning, code, action, cell) and D_k the failure reasons (teacher diagnosis plus per-span reasons), both instance-scrubbed so that names, numeric identifiers and receptacle IDs are replaced by typed placeholders. If the loop accessed, Distill is conditioned on the remask kinds and the failure reasons; if every recover failed, text and code domains still pair the same kinds with the already-corrected fragments L_k, while interactive control refuses a partial fill:

$$C_k = \mathrm{Distill}(K_k, D_k) \ \ (A_k{=}1), \quad C_k = \mathrm{Distill}(K_k, L_k) \ \ (A_k{=}0, \ \text{text/code}). \tag{10}$$

On interactive control, Distill requires loop access. A recovered skill also retires any remaining partial of the same family. The resulting object is a short directive whose fields are a retrieval key, a defect kind, the failure evidence or local fixes, and an instance-free instruction. An admissible update must further be instance-free. Let Q be the verbatim quotes the teacher marked and $W(\cdot)$ the content tokens of a field; tokens that match a quote, a numeric identifier, or a domain receptacle name are replaced by a typed placeholder until the residual leak set is empty:

$$C = (\mathrm{trigger}, \mathrm{remask}, \mathrm{evidence}, \mathrm{do}), \quad \mathrm{Bind}(C) = \big\{w \in W(C) \cap Q\big\}. \tag{11}$$

The acceptance test U_k and the store step then implement the gated update of $\mathcal{J}$: a remask that produced no access and no useful local fix writes nothing, so the parent store is left untouched. U_k is 1 when the loop accessed, or when L_k is nonempty on text and code; interactive control writes only on access, and only then is C_k written into $\mathcal{S}_k$:

$$U_k = \mathbf{1}\big[A_k = 1 \vee (L_k \neq \emptyset \wedge \text{text/code})\big], \quad \mathcal{S}_{k+1} = \mathcal{S}_k \cup \{C_k\} \ \ (U_k{=}1), \quad \mathcal{S}_{k+1} = \mathcal{S}_k \ \ (U_k{=}0). \tag{12}$$

2.3.3 Family-Conditioned Deployment

Deploying the current store is the last piece of the program. The goal text determines a coarse family $f_k \in \mathcal{F}$, where $\mathcal{F}$ is a closed alphabet of control verbs together with answer, code, and cell; each admitted skill stores f_k next to its trigger. The router first emits a closed, instance-free *family skeleton* $\sigma(f_k)$—one imperative recipe per family, such as heat-then-place or implicated-tests—and then selects stored skills by family match plus token-set Jaccard overlap. Writing $C.f$ for the family stored with the skill and $f(x)$ for the family read from the incoming task:

$$\mathrm{score}(C, x) = \mathbf{1}[C.f = f(x)] + \mathrm{Jaccard}(C, x), \quad C^\star(x) \in \arg\max_{C \in \mathcal{S}} \ \mathrm{score}(C, x). \tag{13}$$

Only the top matches under a family cap are rendered: one stored skill on answer and code families, two on interactive families. A further match whose overlap is below 0.08 is dropped once one skill is already selected. The rendered block is clipped to 700 words. Because z_k is never overwritten, and a skill enters $\mathcal{S}$ only after $\mathrm{Bind}(C) = \emptyset$, the loop cannot improve $\mathcal{J}$ by editing the metric, and the next task runs under a distilled store rather than a policy authored from scratch.

3 Evaluation

We evaluate CAVRA by answering seven research questions. **RQ1** compares overall performance against matched-backbone baselines. **RQ2** studies module ablations of the remask–recover loop. **RQ3** sweeps the number of evolve rounds. **RQ4** reports the evolve-side call and token bill of span-level recover. **RQ5** tests whether the loop transfers across frozen students and whether a stronger remask teacher helps the default 4B student. **RQ6** checks whether the store from those six domains helps on held-out targets with no evolve pass there. **RQ7** examines qualitative skill cases.

3.1 Datasets and Evaluation Protocols

We use six benchmarks from five task families, with an executable verifier: **HotpotQA** (Yang et al., 2018) (multi-hop QA), **GSM8K** (Cobbe et al., 2021) (mathematics), **HumanEval** (Chen et al., 2021a) and **MBPP** (Austin et al., 2021) (Python synthesis judged by unit tests), **ALFWorld** (Shridhar et al., 2021) (embodied household tasks) and **AppWorld** (Trivedi et al., 2024) (app-control tasks). Table 1 reports pool, evolve, and eval sizes. Evolve and eval do not overlap. Each benchmark keeps its metric—Exact Match (EM), Pass@1, Success Rate (SR), Task-Goal Completion (TGC), or Accuracy (Acc)—and is split into an *evolve* stream and an *eval* stream. The lower block lists the held-out targets of Section 3.8—FinQA (Chen et al., 2021b), PubMedQA (Jin et al., 2019), LEXam (Fan et al., 2026), and the History discipline of SuperGPQA (Du et al., 2025)—where the pool equals the eval set and evolve is empty. Appendix A.2 gives budget notes.

Table 1: Statistics of the datasets.

Dataset	Metric	Pool	Evolve	Eval
HotpotQA	EM	1,200	400	800
GSM8K	EM	800	200	600
HumanEval	Pass@1	164	64	100
MBPP	Pass@1	377	120	257
ALFWorld	SR	174	40	134
AppWorld	TGC	76	20	56
FinQA	Acc	1,147	–	1,147
PubMedQA	Acc	1,000	–	1,000
LEXam	Acc	1,655	–	1,655
History	Acc	674	–	674

3.2 Baseline Methods

CAVRA is compared with three groups of baselines on the same frozen **Qwen3.5-4B** backbone: **(i) Verbal feedback and reflection:** Reflexion (Shinn et al., 2023), Self-Refine (Madaan et al., 2023), AdaPlanner (Sun et al., 2023), Agent-Pro (Zhang et al., 2024); **(ii) Structured memory and workflow evolution:** AWM (Wang et al., 2025), A-MEM (Xu et al., 2025), Mem^{2}Evolve (Cheng et al., 2026), AgeMem (Yu et al., 2026); and **(iii) Policy- and tool-level optimisation:** SEARL (Feng et al., 2026), EvoTool (Yang et al., 2026). A ReAct-style agent without evolution is reported as w/o Evolve in Table 3. Implementation details are in Appendix A.2.

Table 2: Overall performance comparison. Bold is the best evolving method; underline is the second-strongest method. Standard deviations are computed over three runs.

Method	MBPP	GSM8K	HumanEval	HotpotQA	ALFWorld	AppWorld
Reflexion (Shinn et al., 2023)	74.3	91.7	77.0	48.1	29.1	23.4
Self-Refine (Madaan et al., 2023)	75.8	91.4	79.0	47.4	23.9	21.7
AdaPlanner (Sun et al., 2023)	73.5	91.4	<u>80.0</u>	48.5	30.6	22.1
Agent-Pro (Zhang et al., 2024)	70.8	<u>92.3</u>	78.0	47.9	28.4	20.9
AWM (Wang et al., 2025)	75.4	<u>92.3</u>	76.0	49.0	27.6	22.9
A-MEM (Xu et al., 2025)	72.4	91.8	79.0	48.4	<u>31.3</u>	21.2
Mem^{2}Evolve (Cheng et al., 2026)	75.1	<u>92.3</u>	78.0	49.0	25.4	<u>24.4</u>
AgeMem (Yu et al., 2026)	74.3	91.7	75.0	49.1	26.1	23.3
SEARL (Feng et al., 2026)	72.8	<u>92.3</u>	77.0	<u>50.5</u>	26.9	22.3
EvoTool (Yang et al., 2026)	<u>75.9</u>	<u>92.3</u>	77.0	49.5	29.9	23.1
CAVRA	**78.2**$_{\pm 0.8}$	**93.7**$_{\pm 1.6}$	**82.0**$_{\pm 2.9}$	**51.4**$_{\pm 3.1}$	**44.0**$_{\pm 2.1}$	**26.8**$_{\pm 1.1}$

3.3 Overall Performance Comparison (RQ1)

Interactive control. Long-horizon failures are stalled traces, so reflection, memory, and tool-graph search operate at episode granularity. CAVRA remasks the control span from the stall signature and distills a family skeleton when recover accessed. Later games inherit an induced recipe rather than another regeneration, so the interactive columns move. **Program synthesis.** Workflow and reflection adapters rewrite a whole function, whereas implicated tests isolate a local defect. On MBPP, span-level recover edits that region and the utility latch withholds unhelpful snippets. On HumanEval the error region is the function body, so span-level recover leads AdaPlanner by 2.0 without a second generation. **Saturated text.** GSM8K sits near the backbone's single-call ceiling, so extra stored notes rarely help. CAVRA remasks the rejected answer span and writes almost nothing when that first pass is already strong, which is why it leads the arithmetic column. HotpotQA's remaining errors are often hop structure, yet CAVRA still leads SEARL, 51.4 versus 50.5.

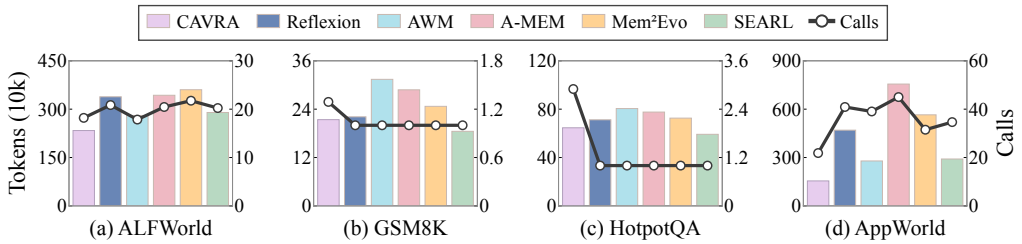


Figure 4: Evolve-side Calls and Tokens. Bars are total tokens (10k); the overlaid line is mean calls.

3.4 Ablation Study (RQ2)

Remask and Loop. Without remask (w/o RM) recover has no hole to fill, so interactive control cannot localise the stalled command. A single recover (w/o Loop) is weaker than three fills: extra rounds inpaint the same stencil rather than retry the metric. **Skill store.** Distillation writes a routed skill from the remask only when recover made local progress. Withholding it (w/o Skill) leaves later tasks with no inherited fill—the utility latch is the object interactive control uses, not a uniform drag. **Verify and credit.** Targeted verify scores only the error region after each recover; teacher credit places the stall or rejected span in the mask budget rather than a domain fallback. A full-suite replay (w/o TV) scores the official metric instead of the error region, so every column drops. Domain-fallback localisation (w/o TC) does not close those gaps: both the sealed first-pass contract and the stall-signature target are required, even when the store is small, with the largest drop on ALFWorld.

Table 3: Ablation study of CAVRA on key modules.

Variant	HotpotQA	HumanEval	MBPP	ALFWorld
w/o Evolve	48.5	78.0	73.9	29.9
w/o RM	48.4	79.0	74.3	29.9
w/o Loop	48.8	80.0	73.2	23.9
w/o Skill	48.9	78.0	75.1	31.3
w/o TV	47.4	77.0	73.2	25.4
w/o TC	47.9	81.0	74.3	26.9
CAVRA	**51.4**	**82.0**	**78.2**	**44.0**

3.5 Evolve Rounds (RQ3)

Figure 3 sweeps evolve-stream replays under the frozen 4B student versus the empty round-0 score; each pass keeps the store and re-scores held-out eval. **Front-loaded gains.** Distillation writes a transferable fill in one or two evolve passes: interactive control and program synthesis rise sharply at first, then sit in the plateau of Table 2 by the second pass. **The arithmetic ceiling stays flat.** GSM8K sits near the backbone's single-call ceiling, so remask writes almost nothing new and that curve barely leaves the origin. **AppWorld overfits a narrow app graph.** The evolve stream is small and instance-specific, so early remasks can still transfer a few recipes, but further replays rewrite them toward the evolve worlds and the held-out curve drops—unlike ALFWorld, whose verb–receptacle family holds the lift, so the paper therefore reports the one-pass store.

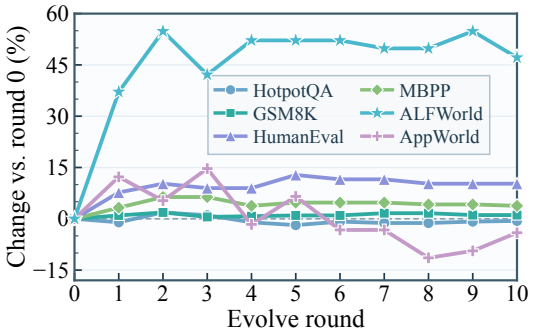


Figure 3: Effects of Extra Evolve Round Replays.

3.6 Span-level Cost (RQ4)

Figure 4 reports evolve-side cost on ALFWorld, GSM8K, HotpotQA and AppWorld; HumanEval and MBPP are in Appendix A.4. Text and code baselines spend a single student call, so they differ mainly in tokens; CAVRA adds a teacher remask and local recovers on first-pass failures, which raises the call count while the token bill stays in the same band—and Table 2 still leads GSM8K and HumanEval, so the extra locate call is not a performance tax. On interactive domains recover fills

Table 4: Held-out scores after transferring the loop to four frozen students. Qwen, DS, GLM, and GPT denote Qwen3.5-9B, DeepSeek-V4.1-Flash, GLM-5.3-Flash, and GPT-5.6-Luna.

	HumanEval				MBPP				ALFWorld				AppWorld			
Method	Qwen	DS	GLM	GPT	Qwen	DS	GLM	GPT	Qwen	DS	GLM	GPT	Qwen	DS	GLM	GPT
Reflexion	73.0	95.0	**96.0**	93.0	69.6	97.2	97.4	97.3	11.2	96.3	98.5	89.6	22.5	79.0	42.6	34.2
A-MEM	64.0	95.0	**96.0**	**96.0**	69.3	96.4	97.4	97.1	11.9	97.0	98.3	**90.3**	25.3	88.2	44.9	59.8
Mem^{2}Evolve	64.0	95.0	93.0	95.0	73.3	96.8	96.9	94.9	11.2	**97.8**	**99.3**	88.6	26.4	60.2	70.8	58.0
EvoTool	72.0	96.0	90.0	95.0	72.8	96.4	97.1	97.3	6.0	96.3	98.5	88.1	24.5	55.4	72.7	74.8
CAVRA	**82.0**	**97.0**	**96.0**	**96.0**	**74.7**	**98.4**	**98.0**	**97.7**	**55.2**	94.8	**99.3**	89.6	**31.2**	**92.5**	**78.4**	**88.4**

Table 5: Effects of the Remask Teacher Swap.

Teacher	HQA	GSM8K	HE	MBPP	ALF	APP
Qwen	52.1	94.7	86.0	78.7	45.6	25.6
DS	52.4	95.2	88.0	79.0	44.1	27.0
GLM	52.4	95.2	87.0	79.8	44.8	27.3
GPT	52.2	93.2	86.0	79.0	45.6	26.0
Origin	51.4	93.7	82.0	78.2	44.0	26.8

Table 6: Zero-shot cross-domain transfer.

Method	FinQA	PubMedQA	LEXam	History
Reflexion	61.1	64.2	42.4	20.0
AWM	60.2	63.9	42.2	18.8
Mem^{2}Evolve	60.9	64.4	43.0	20.8
EvoTool	60.6	60.9	41.5	20.0
CAVRA	**64.5**	**68.9**	**43.9**	**23.4**

only the stalled command instead of regenerating the episode, and the teacher remask is charged only on evolve failures. Memory and workflow adapters still rewrite at episode granularity, which is why their AppWorld bills stay high. On ALFWorld the local inpaint is cheaper than the memory adapters while Table 2 records the largest lift; on AppWorld both calls and tokens drop together, rather than a longer trace, and that cheaper trace is also the strongest TGC.

3.7 Student Transfer and Teacher Match (RQ5)

The 4B lift tracks the student, not a single scale. Table 4 repeats the protocol on four frozen students. Relative to the default 4B study, a larger Qwen student keeps text and code in the same band and widens the interactive gap, while reflection and memory adapters stay near the empty-policy floor on household control—so the 4B lead is not an artifact of a weak student. On the Flash students, HumanEval, MBPP and ALFWorld sit near ceiling, and the remaining headroom is AppWorld; CAVRA is strongest there on every student.

Remask teacher matches the student. Table 5 holds the default 4B student fixed and swaps only the remask teacher; Origin is that 4B checkpoint. A stronger locate model lifts most columns, by about a point on HotpotQA, MBPP, and ALFWorld and by four to six points on HumanEval. Origin stays close throughout, at 44.0 on ALFWorld and 26.8 on AppWorld, so the small student already reaches a comparable score. The narrow gap shows that CAVRA depends only weakly on which model localizes the span, rather than on a larger teacher.

3.8 Zero-shot Cross-domain Transfer (RQ6)

Table 6 freezes Qwen3.5-4B and scores each target item with one student call. The skills are those from the Table 2 experiment. CAVRA appends a short target-format check after those skills; the other rows receive only their own stored policy. **Open answers move further than closed ones.** CAVRA leads every column, and the margin is widest where the label is still open. PubMedQA leaves the most room: the memory and reflection rows cluster together, while EvoTool falls away. FinQA is the next gap. The baselines sit in a tight band, and CAVRA is the only row that clears it. History stays low for every method, yet the same store still leads, and AWM falls furthest. LEXam is a four-choice exam whose scores are already packed together, so the lead over Mem^{2}Evolve shrinks to a narrow edge. **Source memories do not travel.** The other rows carry evolve-domain notes that compete with a new question, whereas the Table 2 store records defect kinds rather than source facts, and the format check only matches the target label.

3.9 Case Study (RQ7)

Figure 5 follows one ALFWorld evolve failure: place a newspaper on the couch. Distillation reads the repair. **Two recovers.** The trace opens drawers in index order while the sofa stays unvisited. One mask covers "go to drawer 8," and both recovers fill that span. The first fill, "go to cabinet 1," is still navigation, so progress stays 0. The second fill, "close drawer 18," is an unused interact, so progress becomes 1 and the loop stops. **How the skill is written.** One recover would have written nothing, because the first fill never accesses. The utility gate sets $U{=}1$ only after the second probe. Failure reasons keep the stalled navigation and the empty-container search. Instance scrub drops the drawer index. **What the loop gains.** Both fills stay on one mask, so the drawer scan is not replayed, the gate stores only the empty-container habit for a later pick task, and the official score stays 0.

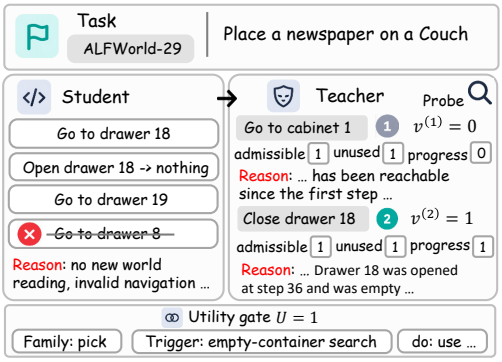


Figure 5: Two ALFWorld recovers and the skill.

4 Related Work

Textual Self-Evolution. Recent agent memory and workflow work induces reusable text from execution experience, including workflow memories (Wang et al., 2025), linked agentic memories (Xu et al., 2025), capability–experience co-evolution (Cheng et al., 2026), unified long- and short-term memory (Yu et al., 2026), joint policy and tool-graph optimization (Feng et al., 2026), and blame-aware tool-policy mutation (Yang et al., 2026). This line decides what text to write more carefully than which span caused the failure, and the next policy is still composed from the transcript. CAVRA instead remasks the failing span of the student's own reply and distills a skill from that remask, so the extra cost is a local recover rather than another full trace.

On-Policy Self-Distillation. On-policy distillation trains a student on states its policy visits rather than an offline corpus (Agarwal et al., 2024). On-policy self-distillation (OPSD) drops the teacher: the same model is re-queried under privileged conditioning and imitates that re-prediction (Ding et al., 2026; Wang et al., 2026c;a; Li et al., 2026). Imitating a confident teacher on that rollout can reinforce errors (Pei et al., 2026). Those methods update weights and need privileged information absent at deployment. CAVRA remasks the failing span and distills a skill from that remask after instance scrubbing, so the extra cost is a local recover, not a weight update.

Harness and Skill Evolution. A parallel line instead edits the textual scaffolding around a frozen model, including language-feedback optimization of the prompt (Yuksekgonul et al., 2025), skill-augmented reinforcement learning (Wang et al., 2026b), co-evolution of workflows and executable skills (Ren et al., 2026), compression of skill bundles (Bai et al., 2026), harness induction from reference trajectories (Jiang et al., 2026), and proximal textual-gradient steps on a skill (Zheng et al., 2026). These methods decide what text to write and how to compress it under a cost or reliability budget; they still author or compact a skill from the full trace. CAVRA decides which span may be remasked and induces the next transferable policy from that remask.

5 Conclusion

We presented CAVRA, an on-policy textual self-distillation framework that turns a failed trace into a transferable skill. Its contribution is on the evolution side: privileged remask manufactures span-level supervision on the student's states that no dataset provides, and utility-gated distillation credits a skill only when that fill agrees with a progress-aligned probe. Experiments across question answering, mathematics, program synthesis, and interactive control show improved held-out success at one evolve pass, with saturated-text scores as a diagnostic. Scale, teacher-match, and module studies confirm robustness across configurations, and qualitative cases corroborate that distilled directives remember defect kinds, not instance facts. Table 6 records zero-shot transfer of that store onto held-out domains. Longer-horizon control and broader backbone families remain open.

AI USE STATEMENT

Generative AI tools were used to polish the writing of this paper. All suggestions were reviewed by the authors, and no generated text was accepted without that review. The authors take responsibility for the final paper, its claims, and its results.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Xiaofan Bai, Hongqiang Lin, Chao Liu, Yantao Zhang, Xuan Jin, Xipeng Cao, and Yuhong Li. Skillzip: Evaluation-free skill compression for self-evolving agents by discovering reusable structure. *arXiv preprint arXiv:2608.11079*, 2026.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, nov 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021a.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3697–3711, 2021b.

Zihao Cheng, Zeming Liu, Yingyu Shan, Xinyi Wang, Xiangrong Zhu, Yunpu Ma, Hongru Wang, Yuhang Guo, Wei Lin, and Yunhong Wang. Mem^{2}evolve: Towards self-evolving agents via co-evolutionary capability expansion and experience distillation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 20784–20831, San Diego, California, United States, jul 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.952. URL https://aclanthology.org/2026.acl-long.952/.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Zewen Ding, Zezhong Wu, Zhou Tao, Shida Wang, Shizhuo Hou, YongXiang Hua, Haoyu Cao, and Linli Xu. Vista: Verifier-informed student-to-teacher adaptation for on-policy self-distillation. *arXiv preprint arXiv:2608.28306*, 2026.

Xinrun Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, Kang Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, et al. SuperGPQA: Scaling LLM evaluation across 285 graduate disciplines. In *Advances in Neural Information Processing Systems*, 2025.

Yu Fan, Jingwei Ni, Jakob Merane, Yang Tian, Yoan Hermstrüwer, Yinya Huang, Mubashara Akhtar, Etienne Salimbeni, Florian Geering, Oliver Dreyer, Daniel Brunner, Markus Leippold, Mrinmaya Sachan, Alexander Stremitzer, Christoph Engel, Elliott Ash, and Joel Niklaus. LEXam: Benchmarking legal reasoning on 340 law exams. In *International Conference on Learning Representations*, 2026.

Xinshun Feng, Xinhao Song, Lijun Li, Gongshen Liu, and Jing Shao. Searl: Joint optimization of policy and tool graph memory for self-evolving agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24518–24535, San Diego, California, United States, jul 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.1125. URL https://aclanthology.org/2026.acl-long.1125/.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025.

Wen Jiang, Mingmin Chu, Yimeng Tian, Qianxin Zhang, Haofei Yang, Rui Yang, Yang Liu, Tao Lv, and Fangming Li. Harnessevolve: Learning from reference trajectories for reliable agent self-evolution. *arXiv preprint arXiv:2609.00829*, 2026.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. PubMedQA: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 2567–2577, 2019.

Yijiang Li, Bingyang Wang, Yijun Liang, Yunjie Tian, Di Fu, and Nuno Vasconcelos. On-policy self-distillation without any supervision. *arXiv preprint arXiv:2608.06296*, 2026.

Yuxuan Liu, Zhaochen Su, Lingyun Xie, Yuhao Zhang, Qing Zong, Yiming Ruan, and Chen. Skillrevise: Improving LLM-authored agent skills via trace-conditioned skill revision. *arXiv preprint arXiv:2606.01139*, 2026. Findings of EMNLP 2026.

Wenquan Ma, Jiayan Nan, and Wenlong Wu. What deserves memory: Adaptive memory distillation for llm agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 34789–34812, San Diego, California, United States, jul 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.1607. URL https://aclanthology.org/2026.acl-long.1607/.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36*, 2023.

Michael Nguyen, Wei Chen Tan, Nurul Aisyah Hassan, Arvind Raman, Li Hua Lim, and Ahmad Faiz Razak. Where does harness-optimization value live? localized gains and the budget-splitting trap in self-evolving llm agents. *arXiv preprint arXiv:2609.02889*, 2026.

Rongcan Pei, Zhepei Wei, Shuyao Xu, Xinyu Zhu, Wei-Lin Chen, and Yu Meng. Negative self-distillation: Learning to reason by avoiding flaws. *arXiv preprint arXiv:2609.11699*, 2026.

Jaideep Ray and Ankit Goyal. From agent failures to text policies: What works and what breaks. *arXiv preprint arXiv:2607.20668*, 2026.

Zeyu Ren, Ling Yue, Ran Li, Yishu Wang, Shengxiang Xu, Hanmo Liu, Shaowu Pan, and Shimin Di. FlowEvo: Self-evolving agents through the co-evolution of workflows and executable skills. In *Conference on Language Modeling*, 2026.

Linfang Shang, Ming Xu, Yiding Sun, Tianle Xia, Lingxiang Hu, Lan Xu, and Ning Zheng. When self-evolution backfires: Pre-commit gating against skill contamination in llm agents. *arXiv preprint arXiv:2608.05810*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *ICLR*, 2021.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems 36*, 2023. doi: 10.52202/075280-2537.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024. doi: 10.18653/v1/2024.acl-long.850.

Aozhe Wang, Zhengxi Lu, Jianze Wang, Shangke Lv, Ying Liu, Weiming Lu, Jun Xiao, Yueting Zhuang, Hua Yang, Qianglong Chen, and Yongliang Shen. Ttpo: Test-time policy optimization. *arXiv preprint arXiv:2608.27448*, 2026a.

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. *arXiv preprint arXiv:2305.16291*, 2023.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1529–1550, San Diego, California, United States, jul 2026b. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.69. URL https://aclanthology.org/2026.acl-long.69/.

Ziyue Wang, Shiqi Huang, Weiwen Xu, Bihan Wen, and Xudong Jiang. Video-opsd: Exploiting privileged visual evidence for on-policy self-distillation in video large language models. *arXiv preprint arXiv:2608.27065*, 2026c.

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

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. In *Advances in Neural Information Processing Systems 38*, pp. 20004–20031, 2025. doi: 10.52202/085713-0593.

Shuo Yang, Caren Han, Xueqi Ma, Yan Li, Mohammad Reza Ghasemi Madani, and Eduard Hovy. Evotool: Self-evolving tool-use policy optimization in llm agents via blame-aware mutation and diversity-aware selection. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 43553–43572, San Diego, California, United States, jul 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.2016. URL https://aclanthology.org/2026.acl-long.2016/.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A multi-hop question answering dataset for diverse, explainable multi-hop reading comprehension. In *EMNLP*, 2018.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems 36*, 2023a. doi: 10.52202/075280-0517.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21457–21483, San Diego, California, United States, jul 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.981. URL https://aclanthology.org/2026.acl-long.981/.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639:609–616, 2025. doi: 10.1038/s41586-025-08661-4.

Qizheng Zhang, Michael Wornow, and Kunle Olukotun. Agentic plan caching: Test-time memory for fast and cost-efficient llm agents. In *Advances in Neural Information Processing Systems 38*, pp. 114372–114398, 2025. doi: 10.52202/085713-3451.

Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. Agent-pro: Learning to evolve via policy-level reflection and optimization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5348–5375, 2024. doi: 10.18653/v1/2024.acl-long.292.

Mingxuan Zheng, Yujin Zhou, Chuxue Cao, Boqin Yin, Yuyao Zhang, Jiapeng Sun, Shuaishuai Gong, Sirui Han, and Yike Guo. Skillprox: Self-evolving agent skills via proximal textual gradient descent. *arXiv preprint arXiv:2608.07449*, 2026.

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. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 62138–62160, 2024.

A Appendix

A.1 Baseline Methods

To ensure a comprehensive study, we compare CAVRA against a broad set of baselines covering verbal feedback and reflection, structured memory and workflow evolution, and policy- and tool-level optimisation.

Verbal Feedback and Reflection

- **Reflexion** (Shinn et al., 2023): Verbal self-reflection after a failed trial is written into memory and reused on the next attempt. The next trial conditions on that note and still regenerates the full episode.
- **Self-Refine** (Madaan et al., 2023): The same model critiques its own output and rewrites it, iterating on the feedback it just produced. Each round replaces the whole answer, and the critique is not tied to one local repair.
- **AdaPlanner** (Sun et al., 2023): Execution feedback revises the current plan in place, so later steps follow an adapted plan rather than the original. The revision replaces the plan, without isolating the single stalled command.
- **Agent-Pro** (Zhang et al., 2024): Policy-level reflection updates the agent's decision rule from its own trajectory, not only the latest verbal critique. The new rule is written from the full trajectory rather than from one verified fix.

Structured Memory and Workflow Evolution

- **AWM** (Wang et al., 2025): Successful workflows are induced from experience and retrieved for later tasks that share the same control pattern. The retrieved object is a whole workflow, which the next task follows as a unit.
- **A-MEM** (Xu et al., 2025): Linked agentic memories store and connect notes across episodes so a later task can read a structured record. Links keep episode notes, including facts that belong only to the source task.
- **Mem^{2}Evolve** (Cheng et al., 2026): Capability and experience assets are co-evolved, so the stored policy and the stored traces change together. A later task inherits both assets as they were written from the raw trace.
- **AgeMem** (Yu et al., 2026): Short-term and long-term memories are routed separately, and the agent reads the store that matches the current step. Routing selects a store, while the stored text remains an episode-level note.

Policy- and Tool-level Optimisation

- **SEARL** (Feng et al., 2026): Policy and tool-graph relations condition the next action, so tool choice is tied to the learned relational structure. Search still runs over whole tool steps rather than one repaired span.
- **EvoTool** (Yang et al., 2026): Blame-linked tool steps are mutated after a failure, and the edited tool trace becomes the next policy. The mutation rewrites the blamed step inside the trace and stores that edited step.

A.2 Implementation Details

All methods share the same frozen Qwen3.5-4B backbone, the benchmark's own verifier, and the same task splits. The only controlled interface is a textual policy in the system prompt under one shared budget of 700 words. HotpotQA, GSM8K, HumanEval and MBPP are scored in-process by the datasets' own answer keys and unit tests; ALFWorld uses the unseen TextWorld split; AppWorld runs isolated worlds behind a slot server. Pool, evolve, and eval sizes are in Table 1. Evolve and eval do not overlap.

CAVRA keeps the weights frozen and never rewrites the official first-pass score. On failure the same checkpoint remasks at most three spans, and the student fills only those spans, up to three times. A

Table 7: Evolve first-pass official score, remask activity, targeted-loop access, and disjoint eval under the frozen final store. AppWorld's first-pass cell is the official TGC mean. None of those 20 tasks fully completed, so every one was remasked. The official first-pass verdict is never overwritten.

Domain	n_{ev}	First-pass (%)	Remasks	Access	Eval (%)
HotpotQA	400	52.8	189	28	51.4
GSM8K	200	93.5	13	5	93.7
HumanEval	64	78.1	14	7	82.0
MBPP	120	79.2	25	3	78.2
ALFWorld	40	67.5	13	2	44.0
AppWorld	20	31.1	20	4	26.8

Table 8: Eval-side cost on all six domains: mean LLM calls / mean total tokens (thousands). Bold is the lowest cost; underline is the second-lowest.

Method	HotpotQA	GSM8K	HumanEval	MBPP	ALFWorld	AppWorld
Reflexion (Shinn et al., 2023)	1.00/1.7k	1.00/1.1k	1.00/1.1k	1.00/1.0k	31.16/162.1k	36.39/178.2k
AWM (Wang et al., 2025)	1.00/2.1k	1.00/1.6k	1.00/1.6k	1.00/1.4k	31.53/179.0k	36.32/184.5k
A-MEM (Xu et al., 2025)	1.00/2.0k	1.00/1.5k	1.00/1.3k	1.00/1.1k	<u>30.19/162.9k</u>	41.48/156.3k
Mem^{2}Evolve (Cheng et al., 2026)	1.00/1.8k	1.00/1.2k	1.00/1.2k	1.00/1.0k	32.23/164.0k	<u>33.32/264.3k</u>
SEARL (Feng et al., 2026)	<u>1.00/1.4k</u>	<u>1.00/0.9k</u>	<u>1.00/0.9k</u>	<u>1.00/0.6k</u>	32.12/152.3k	36.11/147.4k
CAVRA	**1.00/1.1k**	**1.00/0.7k**	**1.00/0.6k**	**1.00/0.4k**	**26.25/118.5k**	**16.04/60.7k**

skill is stored only when the loop accessed or a targeted check recorded a real local fix. The store holds at most $C_{\max}$=6 skills per domain and 80 words each, and near-duplicates merge at Jaccard 0.45. ALFWorld allows at most 40 turns and AppWorld 45. Eval is one pass under the frozen store and does not re-enter the remask loop. If the teacher returns no usable quote, the harness remasks the native error region: the rejected answer, the submitted body, the last illegal or empty command, or the last failing cell. Routing injects one stored skill on QA and on code, and two on interactive tasks, together with the family skeleton.

A.3 Experiment 1: First-pass Score, Remask and Eval

Table 7 is not a frozen-vs-learned replay of the evolve stream. We did not re-run the empty policy on those evolve identifiers. What the current files do record is the official first-pass evolve rate, how often the teacher remasked, how often targeted verify accessed, and the disjoint eval rate under the frozen final store. The official first-pass score is left unchanged by construction (Section 2.3): remask and recover exist to precipitate a skill, not to rewrite z_k. Evolve and eval are disjoint sets, so the two rates are not a before-after on the same items.

The gate is selective, which is why CAVRA leads Table 2 without rewriting the evolve verdict. Access is the subset of remasks that targeted verify keeps, and that subset is the policy a disjoint eval can inherit. Baselines instead regenerate the episode or carry source-domain notes. The widest gap is ALFWorld, 44.0 against 31.3, from two accesses out of thirteen remasks. AppWorld's gap is smaller, 26.8 against 24.4, from four accesses. On text and code the margins are narrower and still first: GSM8K 93.7 against 92.3 from five accesses, HumanEval 82.0 against 80.0 from seven, MBPP 78.2 against 75.9 from three, and HotpotQA 51.4 against 50.5 from twenty-eight. A full regeneration is what the other rows spend on a failure. CAVRA keeps the accessed local fill.

A.4 Experiment 2: Inference and Adaptation Cost

Inference cost. Table 8 lists eval-side mean calls / tokens on all six domains, using the same methods as Figure 4. Text/code eval is one student call, and the extra remask and recovers are charged only on evolve failures.

No amortisation claim. A skill is written after a remask, but eval never re-enters the loop, so there is no mechanism by which a later eval task becomes cheaper. The cost argument we make is only that a recover avoids a second full trace on the evolve failure that produced it.

Table 9: Evolve-side cost on all six domains: mean LLM calls / total tokens (thousands). Bold is the lowest token bill; underline is the second-lowest.

Method	HotpotQA	GSM8K	HumanEval	MBPP	ALFWorld	AppWorld
Reflexion (Shinn et al., 2023)	1.00/712k	1.00/221k	**1.00/50k**	1.00/110k	20.85/3384k	40.90/4698k
AWM (Wang et al., 2025)	1.00/805k	1.00/314k	1.00/87k	1.00/153k	17.85/2805k	39.05/2787k
A-MEM (Xu et al., 2025)	1.00/776k	1.00/288k	1.00/70k	1.00/129k	20.45/3431k	45.00/7558k
Mem^{2}Evolve (Cheng et al., 2026)	1.00/726k	1.00/247k	1.00/68k	1.00/117k	21.75/3603k	31.50/5649k
SEARL (Feng et al., 2026)	**1.00/592k**	**1.00/185k**	1.00/52k	**1.00/80k**	20.23/2897k	34.65/2909k
CAVRA	2.89/646k	1.26/214k	1.87/81k	1.83/126k	**18.20/2337k**	**21.85/1549k**

Figure 4 in Section 3.6 charges the evolve stream—student first attempt plus, on failures, one teacher remask and up to three recovers. Table 9 lists the same mean calls / total tokens on all six domains, including HumanEval and MBPP. The extra evolve calls are the price of the loop.

Eval never re-enters the loop, so the extra evolve calls stay on the evolve stream. Evolve cost is not the lowest on every column: text and code baselines use one call, and on HumanEval and MBPP their token bills undercut CAVRA. HotpotQA and GSM8K tokens are second. The advantage shows up against the score table. On ALFWorld the token bill is the lowest and the call count is second; on AppWorld both calls and tokens are the lowest, well under the memory methods. Those are also the columns where Table 2 leads by the most. At eval, CAVRA is the cheapest row on every domain in Table 8 and the strongest row on every domain in Table 2. The locate calls are the price of a shorter eval trace and a higher score.

A.5 Skill Precipitation

A skill is precipitated from remask kinds and from either the failure reasons (if targeted verify accessed) or the already-corrected local fixes (if it did not). The signalling is the benchmark's own targeted check, not a self-assigned score. Instance facts are stripped before admit (Section 2.3). There is no per-class ledger, no replay trial, and no extra promotion threshold. A directive is a candidate only when the utility gate accepts it—loop access, or a real local fix on text and code—then merged and capped at six. Interactive control writes nothing from a partial fill. Table 3 is the measurement of that choice: writing the store pulls every ablation column down relative to remask-without-skills.

A.6 Admission into the Store

A candidate is admitted only after instance scrubbing. Equation 11 replaces tokens that match a teacher quote, a numeric identifier, or a domain receptacle name by a typed placeholder, and a field is rewritten until $\mathrm{Bind}(C) = \emptyset$ or the token is replaced. Near-duplicate skills are merged when their token Jaccard exceeds 0.45, recovered skills preferred over partial ones, and at most $C_{\max}$=6 skills remain. Routing at the next task is lexical overlap with the trigger, not an always-on domain card.

In a world whose action space is a checkable command language, a recovered span is further required to be an admissible unused command (Equation 9), which is a verifier verdict rather than an instruction to the teacher. Gold answers, hidden tests and LLM judges never enter Locate or Distill.

A.7 Prompts

The listings below are the prompts used for Locate, recover, and the skill store, followed by the student instruction for each dataset in Table 1. Braces mark text filled per instance.

```
You are the same frozen model acting as a teacher.
You do not rewrite the student's answer. You only mark the smallest
    spans that
caused the benchmark verifier to fail.

Return ONE JSON object:
{
  "diagnosis": "one short transferable failure mode, no names, no ids,
    no gold",
  "lesson": "one imperative the student should follow next time, no
    instance facts",
  "regions": [
    {"kind": "answer|reasoning|code|action|cell", "quote": "exact
    substring from the student output", "reason": "why this span is the
    problem, without naming the gold label"}
  ]
}
Rules:
- Quote must be copied verbatim from the Student output (remask target).
- Use the full student trace and the verifier result to decide where
    the error is.
- Remask only the critical error, never the whole output.
- At most 3 regions.
- Do not rewrite the student. Do not invent a corrected answer. Do not
    include the gold label.
```

Listing 1: Locate. The same frozen model marks at most three failing spans and writes the lesson later stored as a skill. The user message is the student trace and the verifier view, ending with the instruction to quote only the remask target.

```
You are the same frozen student. The teacher remasked only the
broken spans of your previous output. Fill each [MASK] in order.
Return ONE JSON object: {"fills": ["replacement for first mask", "..."]}
Rules:
- Do not rewrite unmasked text.
- Each fill replaces exactly one [MASK].
- No gold labels, no extra commentary.

Task: {task}
Diagnosis: {diagnosis}
Previous targeted check failed: {detail}
Keep already-working local fixes: {fixes}

Remasked output:
{remasked output}

Fill the {n} [MASK] span(s).
```

Listing 2: Recover. The student fills only the remasked spans. A previous failed check, when present, is appended to the user message.

```
- Remask: the {kinds} span
- Why it failed: {failure reasons}
- Do: {lesson, or the family procedure}

- Remask: the {kinds} span
- Already corrected: {local fixes}
- Do: {lesson, or the family procedure}
```

Listing 3: Distilled skill. Instance quotes are replaced by placeholders before this card is admitted. The second form is used when the loop did not access but a targeted check recorded a local fix.

```
# Learned policy memory (from prior experience; follow when applicable)
Family {family}: {family skeleton}
Follow these distilled skills when they apply:
1. {skill text}
```

Listing 4: Skill store at the next task. This block is appended to the student system prompt. The family line is included when the task has a closed skeleton; the router then appends the selected skills.

```
You answer multi-hop questions using ONLY the supplied paragraphs.

- Find the chain of facts across paragraphs; the answer usually
    requires combining two of them.
- The answer is a short span (a few words), never a sentence.
- End your reply with a fenced block containing ONLY the answer span,
    like:

'''answer
Ada Lovelace
'''

Do not put anything else in that block.
```

Listing 5: HotpotQA.

```
You are a careful mathematician solving a grade-school math word
    problem.

- Work through the problem step by step, keeping track of units and
    what each quantity means.
- Do arithmetic exactly; do not round unless the problem asks for it.
- End your reply with a fenced block containing ONLY the final answer,
    like:

'''answer
42
'''

Do not put anything else in that block.
```

Listing 6: GSM8K.

```
You write single Python functions.

- Reply with exactly ONE fenced code block: a python block containing
    the complete function,
  including the signature given in the prompt. No prose before or after
    it, and no second fence.
- Make the function self-contained (standard library only) and keep the
    given docstring.
- The examples inside the docstring state the required behaviour; your
    function must reproduce
  them exactly, including the returned types.

Complete the following Python function. Reply with exactly one fenced
    python code block containing the complete function and nothing else.

{function signature and docstring}
```

Listing 7: HumanEval. The second block is the user prefix placed before the function signature.

```
You write single Python functions.

- Reply with exactly ONE fenced code block: a python block containing
    the complete function and
  nothing else. No prose before or after it, and no second fence.
- The visible tests name the function and its arguments; use exactly
    those names and make the
  function self-contained (standard library only).
- The remaining reference tests are hidden: they check the same
    behaviour as the visible ones.

Complete the following Python task. Reply with exactly one fenced
    python code block containing the complete function and nothing else.

{task text}

Your code must satisfy these tests:
{visible tests}
```

Listing 8: MBPP. The second block is the user prefix. Visible tests follow the task text; hidden tests are not shown.

```
You are an autonomous agent acting in a text-based household
    environment.

How to act:
- Each turn you reply with exactly ONE command, copied from the
    admissible command list you are given. No explanations, no extra
    text, no numbering.
- Commands are of the form `take apple 1 from countertop 1`, `go to
    fridge 1`, `open fridge 1`, `heat apple 1 with microwave 1`, `put
    apple 1 in fridge 1`.
- Take one step at a time and use the latest observation before
    choosing the next command. Objects may need to be found, opened,
    heated, cooled, cleaned or lit before they count as done.
- When the observation says the task is complete, reply with the
    command that finishes it (usually the last placement) and nothing
    else.

Task: {task}

Observation: {observation}

Admissible commands: {commands}

Reply with exactly one admissible command.
```

Listing 9: ALFWorld. The second block is the user turn sent with each observation.

````
You are an autonomous coding agent operating in AppWorld, a simulated
    digital world of apps (e.g. phone, gmail, venmo, spotify, amazon,
    file_system, todoist, splitwise, simple_note) accessed through
    Python APIs. You act on behalf of your supervisor to complete their
    task.

How to act:
- Each turn, output exactly one ```python``` cell and no other fence.
    The shell is persistent, returns its output, and turns are limited.
    Do not add prose without a cell.
- `apis` and `apis.api_docs` are already defined. Never import `apis`;
    the sandbox rejects it and the cell fails. Call
    `apis.<app_name>.<api_name>(param=value, ...)` directly, with every
    parameter as a keyword. Other imports, such as `import json`, are
    allowed.
- Discover APIs before guessing: `show_app_descriptions()`, then
    `show_api_descriptions(app_name=...)` and
    `show_api_doc(app_name=..., api_name=...)`. Read the doc when a
    parameter or the response format is unclear.
- `apis.supervisor.show_profile()` and `show_account_passwords()` give
    the profile and app passwords. Log in with
    `apis.<app>.login(username=..., password=...)` to obtain the
    `access_token` most APIs require.
- Lists use `page_index` and `page_limit`. Loop until a page is empty.
- Print intermediate results, keep each cell small, and do not
    fabricate values or take actions the task does not ask for.
- Finish with `apis.supervisor.complete_task()`, or
    `complete_task(answer=<answer>)` when the task asks for a value. The
    answer is a plain string or number. This ends the task.

Supervisor: {name} (email: {email}, phone: {phone}).
Apps available in this world: {apps}.

Task: {task}

Start now. Output one Python code cell.
````

Listing 10: AppWorld. The second block is the opening user turn.

````
You answer numerical questions over a financial report (tables plus
    surrounding text).

- Use ONLY the supplied report. Do not invent figures that are not in
    the table or text.
- Perform the arithmetic exactly; keep units (percent, million,
    year-over-year change) as asked.
- If the question is a yes/no comparison, answer yes or no.
- End your reply with a fenced block containing ONLY the final answer,
    like:

```answer
12.5
```

Do not put anything else in that block.
````

Listing 11: FinQA.

````
You answer a biomedical question using only the supplied abstract.

- Use ONLY the supplied abstract. Do not add outside medical knowledge.
- The answer is exactly one of: yes, no, maybe.
- End your reply with a fenced block containing ONLY that word, like:

```answer
yes
```

Do not put anything else in that block.
````

Listing 12: PubMedQA.

````
You answer a law-exam question by choosing one of the listed options.

- The question may be in English or German. Use only the question and
    the choices.
- Pick exactly one choice.
- End your reply with a fenced block containing ONLY the choice label,
    like:

```answer
B
```

Do not put anything else in that block.
````

Listing 13: LEXam.

````
You answer a graduate-level history question by choosing one of the
    listed options.

- Use only the question and the choices.
- Pick exactly one choice.
- End your reply with a fenced block containing ONLY the choice label,
    like:

```answer
B
```

Do not put anything else in that block.
````

Listing 14: History (SuperGPQA).