SKT: SELF-EVOLVING AGENT SKILL LIBRARY WITH PLAN-DRIVEN TEST-TIME SELF-CORRECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon LLM agents must plan and act over many environment turns, yet adapting them by updating parameters is costly and often impossible behind an API. Inference-time learning from the agent's own trials is the standard alternative, but memory-based methods keep experience as episodic text that is never compiled into procedures, code-based skill libraries apply only where actions are code, and step-level verifiers deliver their verdicts through the prompt, a channel that a small frozen policy can attenuate or ignore. To address this, we propose SKT, which evolves a natural-language procedural skill library from the agent's own successes and failures, consolidates failures across trials into new skills, and seats the same frozen weights in a separate verifier context whose corrections are executed directly as environment actions. Under a matched evaluation protocol with a frozen Qwen2.5-7B-Instruct, SKT consistently outperforms inference-time baselines on held-out ALFWorld tasks, approaches an RL-fine-tuned reference without any weight update, and ports unchanged to multi-hop QA and, on a commercial API backbone, to WebShop.

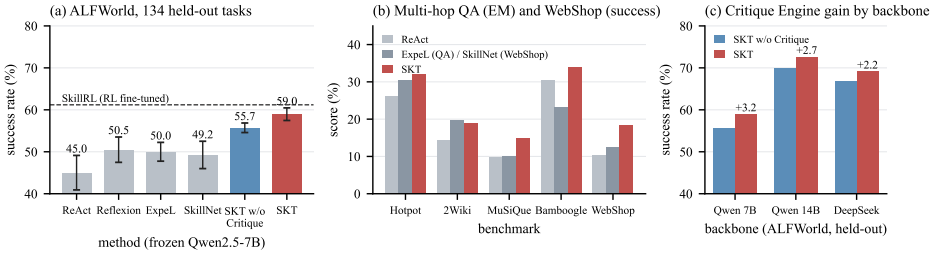


Figure 1: **SKT with a frozen backbone.** (a) Under one matched harness on held-out ALFWorld tasks with a frozen Qwen2.5-7B-Instruct, SKT outperforms every inference-time baseline and approaches the RL-fine-tuned SKILLRL checkpoint, evaluated in the same harness, without a weight update. (b) The same operators port unchanged to multi-hop QA (same 7B model) and to WebShop (all arms on the DeepSeek-V4-Flash API backbone). (c) The plan-driven correction stage adds a consistent gain on three backbones. Bars are means over seeds; whiskers are standard deviations.

1 INTRODUCTION

Language agents that reason, plan, and act over extended interactions with an environment (Yao et al., 2023; Shridhar et al., 2021; Yao et al., 2022) are the standard setting for studying language models under sustained autonomy. Once a base model is fixed, improving it through parameter updates is expensive, often impossible against a closed API, and liable to corrode the general abilities that made the model useful. A complementary line of work therefore lets the agent improve *at inference time* from its own experience (Shinn et al., 2023; Wang et al., 2023; Zhao et al., 2024; Chhikara et al., 2025), and has produced two dominant paradigms. *Memory-based* methods store per-trial reflections or distilled insights and retrieve them into the prompt (Shinn et al., 2023; Zhao et al., 2024; Chhikara et al., 2025). *Code-based skill libraries* compile experience into reusable programs (Wang et al., 2023; Sun et al., 2023; Zhang et al., 2025; Zhou et al., 2024). However, each paradigm carries a structural limitation: (i) memory-based methods keep experience in surface form,

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.

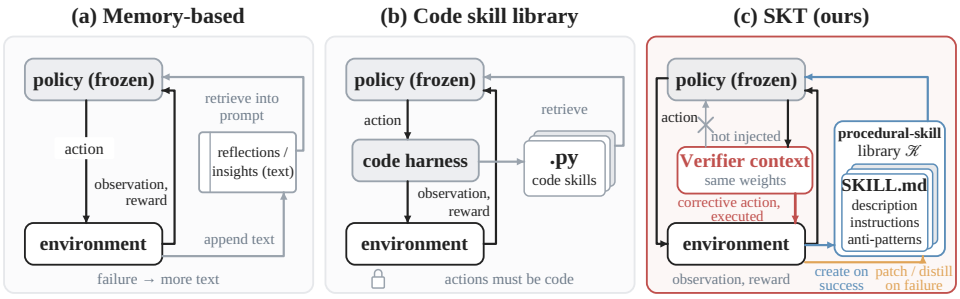


Figure 2: **Three ways to improve a frozen agent from its own trials.** (a) Memory-based methods store reflections or insights and retrieve them into the prompt; experience stays in surface form and failure enters only as more text. (b) Code-based skill libraries compile experience into programs but bind to environments whose actions are code. (c) SKT evolves natural-language procedural skills from both successes and failures, consolidates failures across trials, and corrects the trace from a separate verifier context whose verdict is executed as an environment action rather than injected into the prompt.

so it is never compiled into an executable procedure and failure enters the store only as more text; and (ii) code-based libraries require an executable harness and bind to environments whose actions are code, excluding embodied text benchmarks and web agents.

Step-level verification offers a further remedy by checking the trace as it unfolds (Chen et al., 2024b; Gou et al., 2024; Kim et al., 2025), and would seem to complement either paradigm. Yet in every inference-time variant the verdict is delivered by *prompt injection*: supervisory text is concatenated into the policy's input. On a small frozen backbone this channel is unreliable: in our controlled comparison, the same verdict delivered as advice yields a gain that is not sign-consistent across seeds, whereas executing it yields a gain that is; and in our earlier uncontrolled runs three published prompt-side verifiers each lowered a 7B policy's success. Verification for small frozen agents needs a channel the policy cannot attenuate, not only a better verifier. Figure 2 contrasts the three designs: memory-based and code-based methods both feed experience back through the policy's input, whereas SKT stores experience as procedures the agent can retrieve and corrects the trace through the environment.

To address the central challenge of adapting a frozen agent from its own experience without a code harness and without corrupting its policy prompt, we introduce SKT, a **S**elf-evolving agent s**K**ill library with plan-driven **T**est-time self-correction (Figure 3). SKT couples two engines around a frozen policy. The *Skill Engine* acts across trials and maintains a library of natural-language `SKILL.md` files (Anthropic, 2025), each an ordered procedure with explicit anti-patterns, through four operators that share the policy's backbone: a creator that writes a skill on success, a refiner that patches anti-patterns on failure, a distiller that periodically consolidates buffered failures into new skills, and a retriever that selects skills at trial start. Because the substrate is language rather than code, the same operators serve embodied, web, and question-answering environments, and because failure is consolidated across trials, the library acquires skills such as `early-termination` and `lost-track-of-progress` that no success-triggered extractor can write. The *Critique Engine* acts within a trial: it seats the same frozen weights in a separate, plan-level verifier context that re-reads the plan and the recent trace from outside the policy's context, and when a plan step has not landed, its corrective action is *executed* as the next environment action rather than *suggested* to the policy.

Because inference-time agents are notoriously protocol-sensitive, we pair the framework with a **matched evaluation protocol**: every method is re-run in one harness with the same frozen backbone, decoding, trial budget, seeds, and documented access to in-distribution experience, and claims rest on sign stability and per-seed order statistics. We evaluate SKT on ALFWorld, four multi-hop QA benchmarks, and WebShop, with results highlighted in Figure 1. With a frozen Qwen2.5-7B-Instruct, SKT reaches $58.96 \pm 1.50\%$ on the 134 held-out ALFWorld tasks, its worst seed above the best seed of every inference-time baseline, and approaches the RL-fine-tuned SkillRL without a weight update (§4.2). The same operators port unchanged to multi-hop QA with the same 7B model and to WebShop with a commercial API backbone, and the correction gain replicates on Qwen2.5-14B and that API model (§4.3). Further analyses isolate each mechanism: a learned library of 19

skills outperforms an authored library of 37 under the same retriever and every operator is load-bearing (§4.4), and a single-flag control shows that the correction gain comes from *acting* on the verifier's verdict rather than reading it, at about 3% overhead (§4.5).

Our work makes three key contributions. (1) We present SKT, an inference-time framework that evolves a natural-language procedural skill library from a frozen agent's own successes and failures, including cross-trial failure distillation, and that corrects the agent from a separate verifier context through direct actuation. (2) We establish a matched evaluation protocol for inference-time agents and show, under it, that SKT outperforms memory-based, skill-library, and reflection baselines with the same backbone and experience, approaching an RL-fine-tuned reference. (3) Through controlled ablations across three domains and three backbones, we show that a learned library beats an authored one, that every operator contributes, and that verification helps a small frozen agent when it acts rather than advises.

2 Related Work and Positioning

Table 1: **Positioning.** SKT is the only framework that evolves *procedural* skills in natural language from both success and failure, consolidates failures across trials, and delivers verification by direct actuation, with a frozen open-weight backbone.

Method	Experience	From failure	Consolidates failures	Verification	Backbone
Reflexion	episodic reflections	appended text	✗	prompt	GPT-4 class
ExpeL	insights + exemplars	appended text	insights only	—	GPT-4 class
Voyager	code skills	✗	✗	prompt (critic)	GPT-4
SkillNet	authored NL skills	✗	✗	—	o4-mini
SkillRL	distilled skill bank	via RL	✓ (in training)	—	Qwen2.5-7B (RL-tuned)
ReflAct	—	✗	✗	prompt	GPT-4 class
SKT (ours)	evolved NL procedures	patch + distill	✓	direct actuation	Qwen2.5-7B (frozen)

Memory and reflection. Reflexion (Shinn et al., 2023) appends per-trial verbal reflections to the next prompt; ExpeL (Zhao et al., 2024) extracts a fixed set of insights from a pool of trajectories and retrieves them with successful exemplars; Mem0 (Chhikara et al., 2025) consolidates salient state across sessions. More recent context-evolution methods keep this form: ReasoningBank (Ouyang et al., 2026) distils reasoning strategies from judged successes and failures into retrievable memory items, and Dynamic Cheatsheet (Suzgun et al., 2025) and ACE (Zhang et al., 2026) grow an in-context playbook of strategies. SKT differs in *what* is stored (procedures with ordered instructions and anti-patterns, retrieved per task rather than accumulated in context) and in *how failure enters the store*: ExpeL and ReasoningBank turn a failure into an advisory sentence, whereas the refiner patches the specific procedure that misled the agent and the distiller clusters many failures into a new procedure with its own instructions and anti-patterns.

Skill libraries. Voyager (Wang et al., 2023) grows a library of code policies in Minecraft; AdaPlanner (Sun et al., 2023), AFlow (Zhang et al., 2025), Learn-by-Interact (Su et al., 2025), and PAE (Zhou et al., 2024) extend compositional skills to browser, code, and OS settings; SkillNet (Liang et al., 2026) supplies curated skills (its ALFWorld set is 37 hand-engineered templates) that the agent retrieves but does not rewrite; AutoManual (Chen et al., 2024a) writes rule manuals with a GPT-4-class model; SkillRL (Xia et al., 2026) distils a skill bank from the agent's experience and co-evolves it with the policy inside an RL loop, so its backbone is fine-tuned and its evolution happens at training time. SKT evolves its library at inference time, in language, from a frozen 7B model's own trials, with no weight update.

Verification and self-correction. Self-Debug (Chen et al., 2024b), CRITIC (Gou et al., 2024), ReflAct (Kim et al., 2025), and the Voyager critic verify at inference time; SCoRe (Kumar et al., 2025), AgentRefine (Fu et al., 2025), and SuperCorrect (Yang et al., 2025) learn to self-correct through fine-tuning. The shared inference-time pattern is prompt injection; SKT verifies from a separate context and actuates the verdict directly, which §4.5 shows is the component that carries the gain. Table 1 summarises the positioning.

3 METHOD

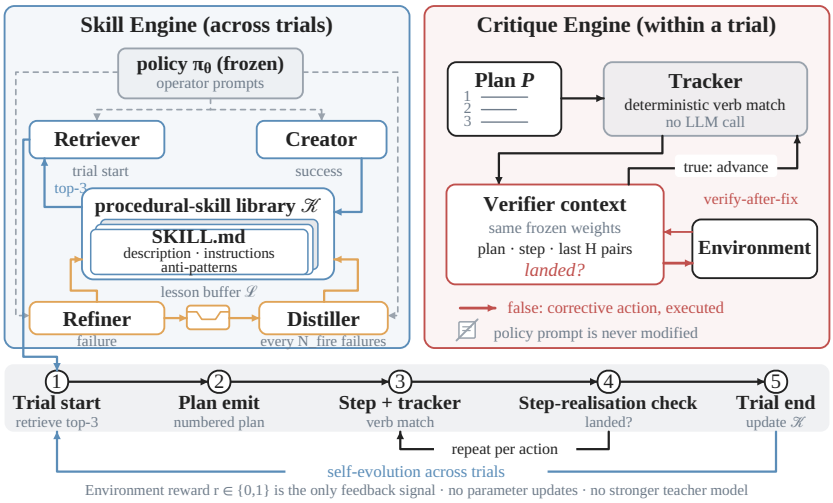


Figure 3: **Overview of SKT.** A frozen policy π_θ is wrapped by two engines. The *Skill Engine* (left) acts across trials: a retriever selects skills at trial start (①); at trial end a creator writes a new `SKILL.md` on success, a refiner patches anti-patterns on failure, and a distiller periodically consolidates buffered failure lessons into new skills (⑤), all writing to the procedural-skill library $\mathcal{K}$. The *Critique Engine* (right) acts within a trial: the policy emits a plan (②), a deterministic tracker detects plan-step boundaries (③), and a separate verifier context, sharing the same frozen weights, judges whether the step landed and, if not, emits a corrective action that is executed directly as the next environment action (④) rather than injected into the policy prompt. The back-arrow from ⑤ to ① closes the self-evolution loop.

SKT couples two engines around a fixed policy (Figure 3); a trial has five stages. ① The retriever returns the top-k=3 skills from $\mathcal{K}$. ② The policy emits a numbered plan $P = (s_1, \ldots, s_M)$. ③ After each action a deterministic tracker verb-matches it against the current step s_c and fires a boundary on match. ④ A verifier context inspects the recent action–observation window and either advances the plan or emits a corrective action that is executed next. ⑤ At trial end the creator or refiner writes to $\mathcal{K}$, and the distiller batch-consolidates buffered failure lessons. Appendix D.2 gives the unified algorithm and a worked example.

3.1 PROBLEM SETTING

We consider environments with text observations and text actions, formalised as a POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, p, T, r)$ with natural-language observations $\mathcal{O}$, a discrete text-action vocabulary $\mathcal{A}$, and a sparse terminal reward $r \in \{0, 1\}$. A trial is one rollout of a policy π_θ until termination or the budget T. Throughout, θ is fixed, modelling the closed-API regime. The environment receives two kinds of actions: policy actions $a_t^\pi \sim \pi_\theta$ and corrective actions a_t^c emitted by the Critique Engine J when the most recent plan step has not realised its intended effect. Across trials, a lesson buffer $\mathcal{L}$ accumulates one reflection per failed trial, and the library $\mathcal{K} = \{(d_k, \alpha_k, \mathrm{instr}_k, \mathrm{anti}_k)\}$ stores skills, each a description, an axis tag, an ordered list of instructions, and an unordered list of anti-patterns.

3.2 SKILL ENGINE: A LIBRARY THAT EVOLVES FROM SUCCESS AND FAILURE

Each skill carries an axis tag drawn from a suggested vocabulary of six types (Information, Planning, Execution, StateTrack, Goal, Termination). The vocabulary is a set of names the operators may assign, not a quota: different domains populate different subsets, and the operators may coin a tag when none fits (§4.4 reports one such emergent tag). Four operators share the policy's backbone and update $\mathcal{K}$ at trial boundaries.

Retriever (trial start) returns the top-k skills by description similarity to the task. **Creator** (trial end, success) decides whether the winning trajectory reveals a reusable pattern and, if so, authors a full `SKILL.md` body in generic class names (*object*, *receptacle*) so that it transfers across instances.

Both are standard patterns (Wang et al., 2023; Zhao et al., 2024). The remaining two operators turn failure into structural updates.

Refiner: per-trial anti-pattern injection. On a failed trial the refiner inspects the trajectory, the failure reason, and the retrieved skills, and decides PATCH or SKIP. A patch appends a one-line anti-pattern (*do not X because Y*) to the most relevant retrieved skill, optionally extending its instructions when a missing step caused the failure; the refiner's reason field becomes the trial's reflection ℓ_i, appended to $\mathcal{L}$. If the failure is attributable to base-model capability rather than skill content, the refiner skips, separating skill bugs from policy bugs.

Distiller: cross-trial consolidation. When $|\mathcal{L}| = N_{\text{fire}}$ the distiller runs one call on the buffered lessons together with a snapshot of $\mathcal{K}$,

$$\mathcal{K} \leftarrow \text{Distill}(\mathcal{L}, \mathcal{K}), \qquad \mathcal{L} \leftarrow \emptyset,$$

clustering lessons by their dominant failure axis, dropping clusters below N_{cluster} members as noise, and for each remaining cluster either refining an overlapping skill or creating a new one. Aggregation is the point: a single failure rarely justifies a structural update, but N_{fire} failures viewed together separate generic procedural patterns from instance noise, yielding skills that no single-trial creator surfaces.

3.3 Critique Engine: plan-driven self-correction from a separate context

The Critique Engine operates within a trial and never writes verification text into the policy's prompt. It composes a plan emitted by the policy, a deterministic tracker, a verifier context J, and direct execution of J's corrective action. The principle is *policy-declared progress, externally verified*: the policy commits to a plan and advances it step by step, and an outside context with no privileged information checks each declared step against the observation stream. Appendix D.2 gives the complete event specification.

Plan and tracker. The first-turn system prompt carries a plan-block directive (Appendix E); the policy emits a numbered plan, parsed into verb-initial steps. After each action the tracker checks whether the action's verb and target match those of s_c; on match a boundary fires. The tracker uses no LLM call and is only a trigger: every boundary it proposes is re-checked by J before any action is taken, so a false trigger costs one verifier call and a missed boundary forgoes at most one correction; neither can produce a wrong action.

Step-realisation check. At each boundary, J receives a compact step-local context (the plan, the current step, the last H action–observation pairs, and the latest action) and returns a fixed-schema JSON with `landed`, a one-sentence reason, an evidence excerpt that must be quoted from the latest observation, and a `corrective_action` when `landed` is false. J has no privileged access: it sees neither the environment state, the admissible-action list, nor the reward, so everything it knows the policy also knows.

Why one backbone can serve both roles. J shares π_θ's weights, yet yields a useful signal where π_θ's own judgement is unreliable, for two structural reasons. *Input asymmetry*: the policy sees the full history, retrieved skills, and prior reflections; J sees a short window not distracted by stale context. *Role asymmetry*: the policy is an open-ended action generator, whereas J is a closed verifier emitting a fixed schema, a much lower-entropy task. Self-correction in SKT is thus one set of frozen weights in two contexts.

Direct actuation and verify-after-fix. When `landed` is true the tracker advances. Otherwise the proposed corrective action is executed as the next environment action; the policy then observes the corrected state as an ordinary action–observation transition in its history and continues from there, with no verdict or advice written into its prompt. A second verifier call confirms the fix (success $\rightarrow$ advance; failed fix $\rightarrow$ next corrective; plan judged wrong $\rightarrow$ replan). Direct execution is chosen for determinism: an executed correction always takes effect, whereas the same verdict delivered as advice can be attenuated or ignored. §4.5 tests exactly this choice.

3.4 Library lifecycle and retry

SKT runs as a three-part pipeline. During *collection*, the policy runs several times on each in-distribution task with both engines active and $\mathcal{K}$ grows from empty; As part of the framework, a task

that fails is re-attempted up to three times, each attempt carrying a one-paragraph reflection on the previous trajectory in the context; a task counts as solved if any attempt succeeds. This reflective retry loop is internal to SKT, and in the experiments every baseline is given the identical trial budget so that no comparison is confounded by it. A deterministic usage/solve-rate filter then *prunes* the library into a frozen $\mathcal{K}_{\text{eval}}$ (Appendix B.1). During *evaluation*, per-trial writes are disabled and the Critique Engine remains active.

4 EXPERIMENTS

4.1 SETUP AND MATCHED EVALUATION PROTOCOL

Benchmarks and backbone. Our primary benchmark is ALFWorld (Shridhar et al., 2021): the library is built on the 140 in-distribution tasks (`valid_seen`) and evaluated on the 134 held-out tasks (`valid_unseen`) spanning six task types. We additionally evaluate multi-hop QA (four benchmarks) and WebShop (§4.3). All methods use Qwen2.5-7B-Instruct (Qwen Team et al., 2025), frozen and served with vLLM (Kwon et al., 2023); §4.3 adds Qwen2.5-14B-Instruct and a commercial API. The frozen ALFWorld library contains 19 skills (16 Creator-origin, 3 Distiller-origin; 16 refined after creation, median 8 versions).

Matched harness. Every inference-time baseline, REACT (Yao et al., 2023), REFLEXION (Shinn et al., 2023), EXPEL (Zhao et al., 2024), and SKILLNET (Liang et al., 2026), is run inside SKT's harness with the identical backbone, greedy decoding, trial budget, and seeds 41–43 (protocol-parity table in Appendix B.2, Table 9). Access to in-distribution experience is documented per method: EXPEL extracts its insights and exemplars from the same 140 trajectories SKT uses for collection (7 insights, a non-degenerate extraction); REFLEXION uses its official prompts; SKILLNET uses its 37 officially released ALFWorld skills through SKT's retriever; REACT uses no experience by design. As a cross-regime reference we include SKILLRL (Xia et al., 2026), a Qwen2.5-7B agent RL-fine-tuned over hand-crafted skills. We evaluate its released checkpoint ourselves, with its official evaluation prompt and skill library, on the same 134 held-out tasks in a single greedy attempt with $T{=}50$, a more conservative protocol than the three-attempt one used for the inference-time rows.

Statistics. We report the mean and sample standard deviation over seeds and, wherever rows share seeds, the paired per-seed difference. With three seeds a significance test has little power, so we anchor claims on two seed-invariant statistics: whether a difference keeps its sign on every seed, and whether per-seed ranges are disjoint. Both are invariant to any reweighting of seeds.

4.2 MAIN RESULTS ON ALFWORLD

Baselines. Table 2 compares SKT with four inference-time baselines under the matched harness: the base policy (REACT), within-episode reflection (REFLEXION), cross-trial insights with exemplar retrieval (EXPEL), and an authored skill library (SKILLNET), together with the RL-fine-tuned SKILLRL checkpoint, evaluated in the same harness, as a cross-regime reference. Table 3 reports the paired per-seed differences down the ladder, and Figure 4(a) visualises the per-seed spread.

Key insights. SKT consistently outperforms every inference-time baseline, and three findings follow that do not rest on a point estimate.

Table 2: **ALFWorld held-out success rate (%) under the matched harness**, per seed and mean±std. Block A re-runs each baseline as its own method. Block B is SKT's own ablation: SKILLNET vs. SKT w/o Critique varies only the library's provenance (37 authored vs. 19 self-evolved, same retriever and k); the last two rows vary only whether plan-driven self-correction is enabled. Δ is the mean gain of full SKT over each row.

	Method	seed 41	seed 42	seed 43	mean ± std	Δ
A	REACT (base policy, shared retry only)	44.78	49.25	41.04	45.02 ± 4.11	↑ 13.9
	REFLEXION (official actor and reflection prompts)	50.00	53.73	47.76	50.50 ± 3.02	↑ 8.5
	EXPEL (insights + exemplars)	52.24	50.00	47.76	50.00 ± 2.24	↑ 9.0
	SKILLNET (37 authored skills, our retriever)	52.99	47.76	47.01	49.25 ± 3.26	↑ 9.7
B	SKT w/o Critique Engine (19 self-evolved skills)	54.48	55.97	56.72	55.72 ± 1.14	↑ 3.2
	SKT (full)	**57.46**	**58.96**	**60.45**	**58.96 ± 1.50**	—
	SKILLRL (RL fine-tuned; released checkpoint, official prompt and skills, single greedy attempt)			61.19		↓ 2.2

The ranking is seed-independent. Full SKT's worst seed exceeds the best seed of every baseline, and the library-only configuration clears the same bar; the per-seed distributions are disjoint, so the

Table 3: **Paired per-seed differences (pp)** down the ladder of Table 2. Each contrast shares seeds and harness; "sign" counts seeds on which the difference is positive.

Contrast	Isolates	seed 41	seed 42	seed 43	mean ± std	sign
SKT w/o Critique − REACT	self-evolved library vs. no experience	+9.70	+6.72	+15.68	+10.70 ± 4.56	3/3
SKT w/o Critique − SKILLNET	learned vs. authored library, same retriever	+1.49	+8.21	+9.71	+6.47 ± 4.40	3/3
SKT − SKT w/o Critique	plan-driven self-correction	+2.98	+2.99	+3.73	+3.23 ± 0.43	3/3
SKT − EXPEL	full framework vs. strongest matched baseline	+5.22	+8.96	+12.69	+8.96 ± 3.73	3/3

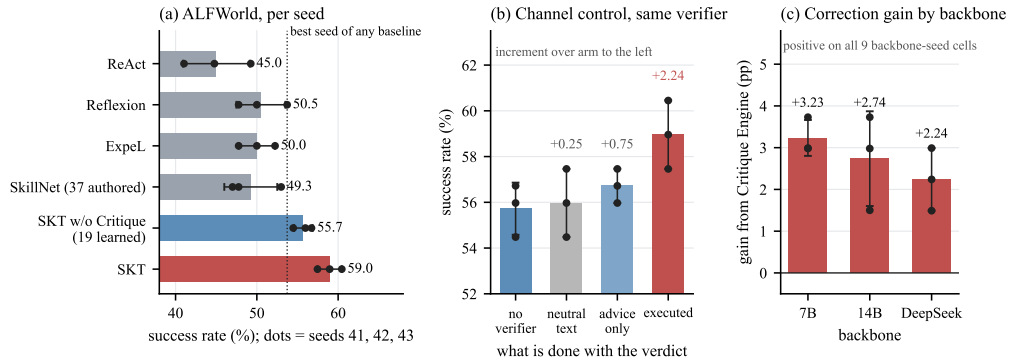


Figure 4: **Per-seed evidence for the three mechanisms** (ALFWorld held-out tasks; bars are seed means, whiskers sample standard deviations, dots individual seeds). (a) Under the matched harness, the worst seed of SKT exceeds the best seed of every baseline. (b) Channel control: the same verifier fires on the same schedule in every arm, and only what is done with its verdict changes; the increment printed above each bar is over the arm to its left. (c) The Critique Engine's gain is positive on every seed of every backbone.

ordering survives any reweighting of seeds. Notably, against EXPEL, the strongest matched baseline and the one given the identical in-distribution experience, SKT gains $+8.96\,pp$ on average with a positive paired difference on every seed (Table 3).

Both mechanisms contribute, and both keep their sign on every seed. The self-evolved library and the correction stage are each positive on all three seeds (Table 3); of the total $+13.94\,pp$ gain over REACT, roughly three quarters come from the library and one quarter from the correction stage. The wider spread of the library contrast is inherited from the REACT anchor, whose seed variance is the largest in Table 2, whereas the two SKT configurations are the tightest rows in the table.

A learned library beats an authored one under the same retriever. Holding retriever, harness, and top-k fixed and swapping only the library's provenance, 19 self-evolved skills outperform SKILLNET's 37 authored skills on every seed, and outperform EXPEL given the same trajectories: a smaller library that was *learned* beats a larger one that was *written*, and §4.4 shows that its 19 entries are 19 distinct procedures. The comparison does not disadvantage SKILLNET: in its own pipeline with the same backbone and a comparable three-run budget it scores 40.30 (Appendix C.1), below the 49.25 it reaches through our retriever. SKILLRL reaches 61.19% in a single greedy attempt only after RL fine-tuning of the same backbone over hand-crafted skills, which SKT approaches with no weight update. Per-type behaviour and trace-level case studies are given in Appendices C.1 and F.

4.3 GENERALITY ACROSS DOMAINS AND BACKBONES

Generalization across domains. We port the unmodified Skill and Critique Engine from embodied ALFWorld to two further settings. On multi-hop QA in the Search-R1 setting (Jin et al., 2025), with the same frozen 7B model as policy and verifier, each benchmark (Yang et al., 2018; Ho et al., 2020; Trivedi et al., 2022; Press et al., 2023) is split per seed into a collection pool and an evaluation pool (Table 4). SKT improves over REACT on all four benchmarks, with disjoint ± 1 std intervals on the three large ones, and matches or beats EXPEL: on MuSiQue, the deepest-hop benchmark, it leads with zero per-seed overlap, it matches EXPEL on HotpotQA, and trails it on 2Wiki. On WebShop (Yao et al., 2022), a third interaction modality, all three arms run end-to-end on the DeepSeek-V4-Flash API backbone of Table 6 rather than on the 7B model (Appendix B.1); the arms separate cleanly at the per-seed level (Table 5): the self-evolved library outperforms both

the no-library and the authored-library control. Notably, the QA libraries populate five of the six schema axes where ALFWorld populates three, consistent with QA requiring evidence collection and intermediate-answer tracking that household tasks do not; the library adapts its content to the domain rather than transferring a fixed set of skills.

Table 4: **Multi-hop QA**, exact match (%) over 5 seeds, same frozen 7B policy and verifier, Search-R1 setting.

EM (%)	SKT	EXPEL	REACT
HotpotQA	**32.02 ± 0.50**	30.53 ± 0.56	26.13 ± 1.26
2WikiMultiHopQA	18.83 ± 1.23	**19.79 ± 0.86**	14.39 ± 0.29
MuSiQue	**14.93 ± 0.65**	10.05 ± 0.88	9.67 ± 0.63
Bamboogle	**33.94 ± 7.16**	23.23 ± 4.36	30.32 ± 6.29

Table 5: **WebShop** library control (test-500, strict success, 3 seeds; all arms on the DeepSeek-V4-Flash API backbone).

Arm	Success (%)
REACT (no library)	10.27 ± 0.12
SKILLNET (official library)	12.60 ± 0.20
SKT (self-evolved)	**18.27 ± 0.70**

Scaling across backbones. Table 6 and Figure 4(c) run the full pipeline on Qwen2.5-14B-Instruct and on a commercial API model, where the entire pipeline executes without local hosting. SKT improves with backbone capability, and the correction stage remains positive on all nine backbone-by-seed cells. Its absolute gain narrows slightly as the base model improves, but normalised by the error that remains without it, the stage removes 7–9% of residual error on every backbone with no monotone trend; plan-driven self-correction therefore does not become redundant as capability grows. One qualification: on 14B the worst full seed coincides with the best library-only seed, so that contrast rests on the paired statistic.

Table 6: **Cross-backbone replication on ALFWorld** (held-out tasks, 3 seeds). Δ is the paired per-seed gain of the Critique Engine; "residual removed" normalises it by the error remaining without it.

Backbone	Configuration	seed 41	seed 42	seed 43	mean ± std
Qwen2.5-7B-Instruct	SKT w/o Critique	54.48	55.97	56.72	55.72 ± 1.14
	SKT	57.46	58.96	60.45	58.96 ± 1.50
	Δ (residual removed 7.3%)	+2.98	+2.99	+3.73	+3.23 ± 0.43
Qwen2.5-14B-Instruct	SKT w/o Critique	69.40	69.40	70.90	69.90 ± 0.86
	SKT	73.13	70.90	73.88	72.64 ± 1.55
	Δ (residual removed 9.1%)	+3.73	+1.50	+2.98	+2.74 ± 1.13
DeepSeek-V4-Flash (API)	SKT w/o Critique	67.16	67.16	66.42	66.92 ± 0.43
	SKT	69.40	70.15	67.91	69.15 ± 1.14
	Δ (residual removed 6.8%)	+2.24	+2.99	+1.49	+2.24 ± 0.75

4.4 SKILL ENGINE ANALYSIS

Every operator is load-bearing. Removing any one operator and rebuilding the library lowers success on all three seeds, and the best seed of each ablated configuration falls below the worst seed of the full system (Table 7). The ordering Creator > Distiller > Refiner is seed-independent and consistent with the operators' roles: the Creator is the only source of skills from successful trajectories, so removing it also starves the other two. The Distiller's cost is the controlled evidence for our second contribution: cross-trial failure consolidation adds content that neither the success-triggered Creator nor the single-trial Refiner supplies.

Table 7: **Leave-one-out ablation of the operators** (ALFWorld held-out tasks, 3 seeds; the library is rebuilt without the operator). Δ is the paired per-seed mean against full SKT; every operator is negative on all three seeds, and the best seed of each ablated configuration is below the worst seed of the full system.

Configuration	seed 41	seed 42	seed 43	mean ± std	Δ vs. full
SKT (full)	57.46	58.96	60.45	58.96 ± 1.50	—
w/o Creator	48.51	47.01	48.51	48.01 ± 0.87	−10.95 ± 1.73
w/o Distiller	49.25	54.48	55.97	53.23 ± 3.53	−5.73 ± 2.15
w/o Refiner	53.73	54.48	56.72	54.98 ± 1.56	−3.98 ± 0.43

The library is used and is not padded. Every skill is retrieved during collection, the most retrieved skills are Creator-origin and each serves one task family, and the three Distiller skills are the only cross-family entries: failure-derived skills encode broad constraints, success-derived skills narrow

procedures. Redundancy tested lexically, semantically, and behaviourally finds no near-duplicate pair, so the 19 entries are 19 distinct procedures; the 19-versus-37 comparison rests on provenance, not on a threshold-dependent count (see Appendix C.2, Table 13).

The schema is a vocabulary, not a quota. ALFWorld's library uses three of the six axes (Planning 10, Execution 7, Termination 2); the QA libraries use five. The QA libraries also contain eight skills (2.6% of tags) under an axis the model wrote itself, *Verification*, covering the check that an intermediate answer actually holds. All eight fall in this one category rather than scattering as noise: the operators use the schema by meaning, not as a label sheet.

4.5 Critique Engine analysis

Acting on the verdict, not reading it, carries the gain. A verifier's benefit could in principle come from extra prompt content, from the judge prompt's design, or from implementation details. Our same-signal control (Figure 4(b); per-seed table in Appendix D.2, Table 15) holds the verifier, its context, and its firing schedule fixed and varies, in a single flag, only what is done with the verdict: replaced by equal-length neutral text, shown to the policy as advice, or executed. Neutral text and advice contribute $+0.25$ and $+0.75\,pp$, neither sign-consistent across seeds; executing the verdict contributes $+2.24\,pp$, roughly 70% of the total and the only component positive on every seed. For a small frozen agent, verification helps *consistently* only when it *acts*. An earlier single-seed comparison in the same harness points the same way: three published prompt-side verifiers (ReflAct, a Voyager-style critic, a Self-Debug/CRITIC-style step verifier) each lowered the library-only baseline while the Critique Engine raised it (Table 16); as they differ from ours in more than the channel, that comparison is motivation only.

The stage is cheap, precise where it fires, and bounded by design. The verifier fires only at plan-step boundaries and usually concludes that no correction is needed; over one run of the 134 held-out tasks it adds about 3% wall-clock and 0.37M tokens on the same frozen backbone (Table 8). Re-examining every corrective action on the three fresh seeds, a quarter to a third achieve the condition they target (strict and lenient rules for repeated examinations); the rest are mostly redundant re-navigation to an already-satisfied state or commands the environment rejects, which do not worsen the state but do consume budget, and the $+3.23\,pp$ of Table 3 is the net effect after that cost (taxonomy in Appendix D.2). The tracker parses every plan and detects boundaries with precision and recall above 96% on a blinded, human-labelled sample; since every trigger is re-checked by the verifier, tracker noise bounds efficiency, not correctness. What the stage cannot reach is also clear: the unsolved trials fail by *exploration*, wandering before locating the target object, which is not a wrong plan step; widening the verifier's reach over plan steps from 36% to 75% in a diagnostic run left success unchanged, so the exploration policy, not verification coverage, is the remaining bottleneck.

Table 8: **Cost and reliability of the Critique Engine**, per run over the 134 held-out tasks (frozen 7B).

Quantity	Value	Quantity	Value
Verifier tokens (input / output)	0.33M / 0.04M	Corrections achieving their target	25–33%
Wall-clock overhead	$+3.1\%$	Plan parsing failures	0 / 805
Same tokens at DeepSeek-V4-Flash list price	$\approx$ \$0.06	Boundary precision / recall	98.0% / 96.1%

5 Conclusion

We presented SKT, an inference-time framework that adapts a frozen policy through a self-evolving procedural skill library, cross-trial failure distillation, and plan-driven self-correction from a separate verifier context whose verdicts are executed rather than suggested. Under a matched evaluation protocol, SKT with a frozen Qwen2.5-7B-Instruct consistently outperforms memory-based, skill-library, and reflection baselines on held-out ALFWorld tasks, approaches an RL-fine-tuned reference without a weight update, and ports unchanged to multi-hop QA and WebShop. Controlled analyses confirm that each mechanism is load-bearing: a learned library beats an authored one under the same retriever, every operator contributes, and the correction gain comes from acting on the verifier's verdict, on all three backbones. The residual failures are exploration failures beyond any plan-step corrector; world-model-guided exploration (Qiao et al., 2024) is the natural next step.

AI USE STATEMENT

In accordance with the conference guidelines, we acknowledge the use of large language models (LLMs) during the preparation of this manuscript. We used LLMs solely as writing-assistance tools, for language polishing and for improving the clarity and readability of the text, including grammar and academic style. All core research contributions, including the methodology, the experimental design, the implementation, the experiments, and the interpretation of results, were developed and carried out by the authors; LLMs did not contribute to the conceptual development, the experimental validation, or the analysis of results, and none of the tasks that require disclosure under the ICLR 2027 AI policy apply to this work. Separately, a frozen LLM is the object of study: the same backbone serves as the agent's policy, its skill operators, and its verifier, and every prompt it receives is printed in Appendix E. We have reviewed all LLM-polished text and take responsibility for the final content of this work.

ETHICS STATEMENT

This work studies inference-time adaptation of frozen, publicly available models on public benchmarks (ALFWorld, WebShop, multi-hop QA). It involves no human-subject data beyond a small boundary-annotation set labelled by one of the authors, releases no new model weights, and does not raise the capability ceiling of the underlying policy. We do not foresee societal risks beyond those already attached to the public backbones.

REPRODUCIBILITY STATEMENT

Appendix B.1 specifies the collection, pruning, and evaluation pipeline, every engine threshold and sampling temperature, and the serving configuration; Appendix B.2 documents each baseline's matched-harness configuration; Appendix E prints the verbatim operator prompts, verifier prompts, plan directive, and policy template. We will release the framework code, the prompts, the frozen evaluation library, and the per-seed run artifacts behind every table.

REFERENCES

Anthropic. Equipping agents for the real world with agent skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, 2025.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Constructing instruction manuals by LLM agents via interactive environmental learning. *arXiv preprint arXiv:2405.16247*, 2024a.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations (ICLR)*, 2024b.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma Gongque, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. Agentrefine: Enhancing agent generalization through refinement tuning. *arXiv preprint arXiv:2501.01702*, 2025.

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 (ICLR)*, 2024.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, 2020.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Conference on Language Modeling (COLM)*, 2025.

Jeonghye Kim, Sojeong Rhee, Minbeom Kim, Dohyung Kim, Sangmook Lee, Youngchul Sung, and Kyomin Jung. ReflAct: World-grounded decision making in LLM agents via goal-state reflection. *arXiv preprint arXiv:2505.15182*, 2025.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023.

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Yida Xue, Xin Xu, Tongtong Wu, Kun Wang, Yang Liu, Zhen Bi, Jungang Lou, Yuchen Eleanor Jiang, Hangcheng Zhu, Gang Yu, Haiwen Hong, Longtao Huang, Hui Xue, Chenxi Wang, Yijun Wang, Zifei Shan, Xi Chen, Zhaopeng Tu, Feiyu Xiong, Xin Xie, Peng Zhang, Zhengke Gui, Lei Liang, Jun Zhou, Chiyu Wu, Jin Shang, Yu Gong, Junyu Lin, Changliang Xu, Hongjie Deng, Wen Zhang, Keyan Ding, Qiang Zhang, Fei Huang, Ningyu Zhang, Jeff Z. Pan, Guilin Qi, Haofen Wang, and Huajun Chen. SkillNet: Create, evaluate, and connect AI skills. *arXiv preprint arXiv:2603.04448*, 2026.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations (ICLR)*, 2026.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Agent planning with world knowledge model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 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 *International Conference on Learning Representations (ICLR)*, 2021.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Ö. Arık. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. *arXiv preprint arXiv:2501.10893*, 2025.

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 (NeurIPS)*, 2023.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10, 2022.

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.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Minkai Xu, Joseph E. Gonzalez, Bin Cui, and Shuicheng Yan. Supercorrect: Advancing small LLM reasoning with thought template distillation and self-correction. In *International Conference on Learning Representations (ICLR)*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

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

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations (ICLR)*, 2025.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations (ICLR)*, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Erran Li. Proposer-agent-evaluator (PAE): Autonomous skill discovery for foundation model internet agents. *arXiv preprint arXiv:2412.13194*, 2024.

A LIMITATIONS

Our evidence rests on three seeds per configuration on ALFWorld (five on QA); with $n{=}3$ we anchor claims on sign stability and per-seed order statistics rather than p-values. The cross-domain results show that the operators and the substrate port unchanged and build a useful per-domain library; they do not show transfer of one learned library across domains, which the design does not attempt. The Critique Engine targets plan-step misses by construction; the residual ALFWorld failures are exploration failures outside its reach. The boundary-annotation set was labelled by one of the authors, blinded to the tracker and verifier decisions, with a second author reviewing a random subset. The verifier is deliberately denied the admissible-action list so that its information never exceeds the policy's; supplying it would remove the rejected-action class of corrections. We evaluate embodied household tasks, web shopping, and multi-hop QA.

B EXPERIMENTAL DETAILS

B.1 EVALUATION PROTOCOL AND HYPERPARAMETERS

Pipeline. *Collection*: the policy runs on the 140 in-distribution tasks with the Skill Engine updating $\mathcal{K}$ and the Critique Engine intervening; the library starts empty. *Prune*: let u_k and σ_k be the usage count and solve rate of skill s_k and $\bar{\sigma}_k$ the usage-weighted mean solve rate of comparable used skills; the frozen library removes $D = \{s_k : u_k = 0\} \cup \{s_k : u_k \geq u_{\min} \wedge \sigma_k < \bar{\sigma}_k - \delta\}$ with $u_{\min}$=5, δ=0.1. *Evaluation*: with the library frozen and per-trial writes disabled, each of the 134 held-out tasks is attempted as described in §3.4. The Critique Engine remains active throughout. All baselines in Table 2 receive the identical three-attempt budget with the same reflection mechanism.

Hyperparameters. Environment budget T=50 actions per attempt; up to 3 attempts per task; top-k retrieval k=3; distillation trigger N_{fire}=20; cluster minimum N_{cluster}=3; verifier history depth H=5. The policy decodes greedily (τ=0). Auxiliary calls: retriever τ=0.1 (256 tokens), creator 0.1 (1024), refiner 0.1 (1024), distiller 0.2 (4500), reflection generation 0.3 (300), verifier 0.0 (256); top-p 0.9 for the operators, 0.95 for reflections, 1.0 otherwise. Qwen2.5-7B-Instruct and Qwen2.5-14B-Instruct are served with vLLM (`bfloat16`, `gpu-memory-utilization` 0.85, `max-model-len` 32768) on NVIDIA A6000 GPUs; the API backbone is DeepSeek-V4-Flash in non-thinking mode.

Seeds and statistics. The policy decodes greedily, so seeds 41–43 do not vary decoding; they set the task order across the environment workers and the sampling of the auxiliary operator calls (Appendix B.1), and the residual spread also includes inference non-determinism of batched serving. The library is rebuilt per seed for every configuration in Tables 7 and 6; the evaluation rows of Tables 2 and 15 use one frozen library per configuration across the three seeds. All $\pm$ values are sample standard deviations (ddof=1) over seeds. The single canonical run of our original submission scored 59.70%, inside the observed per-seed range $[57.46, 60.45]$ and within one standard deviation of 58.96 ± 1.50.

Multi-hop QA. HotpotQA, 2WikiMultiHopQA, MuSiQue (dev sets) and Bamboogle (test set) are shuffled with a per-seed generator and split 50/50 into a collection pool, on which the library is built, and an evaluation pool. Retrieval uses the Search-R1 Wikipedia-2018 corpus with an e5-base-v2 dense index; evaluation decodes greedily and scores exact match with the Search-R1 normalisation; seeds 41–45.

WebShop. The library is built on a stratified sample of 500 training sessions disjoint from the canonical test-500 split; evaluation uses the same three-attempt budget over test-500 with a 15-action budget per attempt and scores strict success (reward $= 1.0$); three seeds; the policy, verifier, and reflections all run on the API backbone.

B.2 BASELINE CONFIGURATION IN THE MATCHED HARNESS

Table 9: **Protocol parity.** Backbone, decoding, trial budget, action budget, and implementation are identical across the inference-time rows. The RL-fine-tuned SKILLRL checkpoint is evaluated with its official prompt and skill library in a single greedy attempt on the same tasks and action budget; the last column records each method's designed experience source.

Method	Backbone	Trial budget	Actions	Seeds	In-distribution experience
SKT	7B frozen	3 attempts	T=50	41–43	library built from the 140 ID tasks
SKT w/o Critique Engine	✓	✓	✓	✓	same library; correction disabled
EXPEL (rerun)	✓	✓	✓	✓	same 140 ID tasks (insights + exemplars)
REFLEXION (official prompts)	✓	✓	✓	✓	within-episode reflections only
SKILLNET (official library)	✓	✓	✓	✓	37 authored skills, no ID exposure
REACT	✓	✓	✓	✓	none
SKILLRL (released checkpoint)	7B RL-tuned	1 attempt, greedy	✓	single run	RL on ID trajectories; official prompt and skill library

All baselines run inside SKT's harness with the same frozen backbone, greedy decoding, three-attempt budget, T=50, and seeds 41–43. **REACT**: bare policy prompt, no retrieved experience. **REFLEXION**: the official Reflexion prompts, with within-episode reflections carried across attempts. **EXPEL**: the official three-stage pipeline (reflection cache; insight extraction with `max_num_rules`=10; evaluation with task-similarity exemplar retrieval), with insights extracted from the same 140 in-distribution trajectories that SKT uses for collection, yielding 7 insights, and exemplars drawn from the same pool. **SKILLNET**: the 37 officially released ALFWorld skills, injected through SKT's retriever with k=3 and otherwise identical prompt construction. **SKILLRL**: the released Alfworld-7B-RL checkpoint, evaluated with SkillRL's official evaluation prompt and its official hand-crafted skill library on the same 134 held-out tasks, greedy decoding, T=50, a single attempt per task, one run. The 61.19% in Table 2 is our own measurement under this single-attempt protocol.

C Additional Results

C.1 Per-task-type results and submission-era reproductions

Table 10: ALFWorld success rate (%) per task type on a single SKT run (59.70% held-out; 56.4% in-distribution), a separate run from seeds 41–43 whose score lies inside their range, and the RL-trained reference; per-type orderings from this single run are diagnostic, not seed-aggregated. Held-out task counts: Pick 24, Look 18, Clean 31, Heat 23, Cool 21, Pick2 17; in-distribution: 35, 13, 27, 16, 25, 24. SkillRL held-out numbers are our matched-harness evaluation of the released checkpoint; its in-distribution row is taken from Xia et al. (2026).

	In-distribution (`valid_seen`, 140)							Held-out (`valid_unseen`, 134)						
Method	Pick	Look	Clean	Heat	Cool	Pick2	All	Pick	Look	Clean	Heat	Cool	Pick2	All
SkillRL (RL)	97.9	71.4	90.0	90.0	95.5	87.5	89.9	70.6	57.9	44.4	57.1	90.5	60.0	61.19
SKT (single run)	71.4	46.2	59.3	81.3	52.0	25.0	56.4	54.2	83.3	67.7	65.2	57.1	23.5	59.70

On this run SKT leads on the multi-step modify-then-place types (clean, heat, cool), which require the coordinated state tracking that retrieved procedural skills encode (`clean-object-and-place-in-receptacle` enforces pick $\rightarrow$ sink $\rightarrow$ clean $\rightarrow$ take $\rightarrow$ place), and on clean and heat it exceeds the RL-trained reference. It does not lead everywhere: the single-action *look-at* type is nearly saturated by any retry mechanism, and on *pick-two* (two coordinated pick-place sequences) parametric training visibly outpaces inference-time methods on a 7B backbone (60.0 vs. 23.5).

For transparency, our submission-era reproductions of the baselines in their *own* code bases with the same frozen backbone, scored on the same 134 held-out tasks as a task solved in any of three independent runs, gave 13.43% (ReAct), 16.42% (Reflexion, with a 20-action budget), 11.94% (ExpeL), and 40.30% (SkillNet). These numbers mix repositories, budgets, and prompt formats, and we use them for no claim; the matched-harness Table 2 supersedes them. The three-to-four-fold gap between the two sets is itself a finding: cross-code-base comparisons of inference-time agents are dominated by protocol, which is why every claim in this paper rests on the matched harness.

C.2 SKILL LIBRARY STATISTICS

Table 11: Retrieval frequency of each of the 19 frozen skills during the collection phase of the single run of Table 10 (the frozen evaluation does not write usage counts back) and the success rate of trials in which the skill was retrieved. Counts are from a single run and correlational; causal attribution is Table 7.

Skill	Origin	Retrieved in N trials (of 140)	Success when retrieved
examine-multiple-locations	Creator	20	0.350
examine-drawers-and-countertops	Creator	18	0.389
cool-object-and-place-in-countertop	Creator	17	0.353
examine-sinkbasin-before-picking	Creator	17	0.353
clean-object-and-place-in-receptacle	Creator	16	0.375
cool-object-and-examine-before-placing	Creator	16	0.313
go-to-shelf-then-toilet	Creator	16	0.563
cool-object-and-place-in-receptacle	Creator	13	0.231
examine-diningtable-before-placing	Creator	13	0.462
incorrect-task-decomposition	Distiller	13	0.308
go-to-sofa-then-coffeetable	Creator	11	0.455
use-light-to-examine-object	Creator	11	0.364
examine-locations-before-finding-object	Creator	9	0.333
early-termination	Distiller	8	0.250
examine-sink-before-cleaning-and-placing	Creator	8	0.375
heat-object-and-place-in-receptacle	Creator	8	0.375
use-desklamp-to-examine-object	Creator	8	0.375
go-to-sidetable-use-desklamp-examine	Creator	5	0.400
lost-track-of-progress	Distiller	3	0.333

Table 12: Axis coverage across the two domains, recomputed from the `SKILL.md` metadata. "off-vocabulary" counts skills whose axis tag lies outside the six suggested names.

Library	Planning	Execution	Information	Goal	StateTrack	Termination	off-vocabulary	axes used
ALFWorld (19 skills)	10	7	0	0	0	2	0	3/6
Multi-hop QA (20 libraries)	181	48	36	25	8	0	8 (Verification)	5/6

Two caveats are reported as measured: the QA engine emits an off-vocabulary *Verification* tag (8 of 306), and a skill's nominal axis field diverges from its birth axis in 2 of 19 ALFWorld cases (the distiller may reassign an axis when its clustering judges the original assignment incoherent with the broader failure pattern). Lexically, the most similar pair of ALFWorld skills reaches ROUGE-L 0.86 and consists of two procedures from the same task family; semantically, embedding cosines (e5-base-v2) have median 0.863 across all pairs, and merging pairs above τ would shrink the library to 9, 17, and 19 skills at $\tau = 0.90$, 0.95, and 0.97; behaviourally, on the logged QA libraries (3,702 trials each) the median pair has co-retrieval Jaccard 0.00 and only the top 1% of pairs co-fire often (p99 0.78–0.97).

Table 13: **Redundancy and usage of the 19-skill library**, over all 171 skill pairs.

View	Statistic
Lexical: ROUGE-L / word-set Jaccard, median pair	0.22 / 0.14
Semantic: e5 cosine, median pair	0.863
Semantic: library size if pairs above τ=0.90 / 0.95 / 0.97 were merged	9 / 17 / 19
Behavioural: co-retrieval Jaccard, median pair	0.00
Usage: trials retrieving each skill, min / median / max	3 / 13 / 20

D MECHANISM DETAILS

D.1 SKILL ENGINE

Self-evolution: three commitments. (i) The evolution target is a structured library whose descriptions, axis tags, instructions, and anti-patterns are all populated by the agent itself, not by templates, gold answers, or curated demonstrations. (ii) The evolution mechanism is four trial-boundary operators sharing the policy's backbone. (iii) The feedback signal is exclusively the trial's binary outcome and its trajectory: no oracle skills, expert demonstrations, or human annotations, and no stronger teacher model, since every LLM call in the framework uses the same frozen backbone. Failure is treated symmetrically with success: the refiner and distiller convert failure trajectories into anti-pattern bullets and structural restructurings rather than discarding them.

Closed-system pipeline. The retriever, creator, and refiner are themselves first-class skills stored in the library's `_meta/` namespace; their bodies are loaded as system prompts that turn the backbone into a dedicated sub-agent for the trial boundary at hand. The distiller runs periodically on a batch of lessons.

Trial-by-trial trace. $\mathcal{K}$ starts empty. The first task arrives; the retriever finds no candidates and the policy runs on its base prompt. After the trial, the creator (on success) or refiner (on failure) examines the trajectory and may CREATE a skill or PATCH a retrieved one; on failure the reflection ℓ_i joins $\mathcal{L}$. From the next task on, the retriever selects up to k=3 skills and splices them into the policy's system prompt. Once $|\mathcal{L}| = N_{\text{fire}}$ the distiller runs once and emits structural refinements. At the end of collection, $\mathcal{K}$ is pruned and frozen.

D.2 CRITIQUE ENGINE: ALGORITHM AND CORRECTION ACCOUNTING

Boundary detection. A pure-Python tracker maintains a current-step pointer c initialised to 1. After each policy action the tracker checks whether the action's verb and target class match those of step s_c; on match it fires a boundary event and, upon verifier confirmation, advances c. Separating detection from verification lets the more expensive verifier be invoked only when the action stream has plausibly progressed.

Verify-after-fix outcomes. After a corrective action is applied and the environment returns a new observation, a second verifier call checks that the intended effect of s_c has now been realised, with three outcomes: confirmed success (advance c, return control to π_θ); failed fix with a new corrective proposed (queued for the next turn); or the verifier judging the plan itself wrong, which escalates to a replan. Any corrective action that itself fails to take effect is therefore observed and acted upon rather than silently absorbed.

When the engine acts: complete event specification. The engine is event-driven and never polls. Exactly one verifier call can occur per environment action, and an intervention requires two gates to agree: a deterministic trigger and a verifier verdict. Table 14 lists every event.

Table 14: Events of the Critique Engine. At most one row fires per environment action.

Event	Condition	Action taken	Cost
Boundary proposed	verb and target of a_t match those of s_c	call the verifier	1 call
Step landed	`landed = true`	$c \leftarrow c + 1$	—
Step missed	`landed = false`, corrective given	execute the corrective as the next action	1 action
Missed, no fix	`landed = false`, no corrective	none; control returns to the policy	—
Verify-after-fix	the previous action was a corrective	call the verifier; advance, next corrective, or escalate	1 call
Escalate	verifier flags the plan as wrong	re-issue the first-turn plan directive; $c \leftarrow 1$	—

Trigger. The trigger is a deterministic comparison of the emitted action's verb and target class with those of the current plan step; an action that does not advance the plan proposes nothing, so the verifier is never invoked while the policy explores. The plan directive asks the policy to state which plan step each action advances; the tracker resolves this declaration from the emitted action rather than from free text, which keeps the trigger deterministic and free of LLM calls.

Gate. A proposed boundary is only a candidate. The verifier must ground its verdict in the latest observation (the `evidence_obs` field is required to be an excerpt of it), and only a `landed = false` verdict with a concrete corrective action changes what the environment receives. A false trigger therefore costs one verifier call; a missed boundary forgoes at most one correction; neither can produce a wrong action.

Bounds. The corrective chain has two terminations: the verifier may judge, on the post-fix observation, that the plan itself is wrong (the target is not where the plan assumed), which escalates to a replan; and the trial budget T bounds every chain. A corrective action the environment rejects ("Nothing happens.") is not silently retried: the rejection is the observation that verify-after-fix reads, after which the verifier proposes a different action, escalates, or returns control to the policy, and every link of the chain consumes one action from T, so the loop between stages ③ and ④ cannot run without bound; what it can do is spend budget, which §4.5 accounts for. Escalation re-issues the same plan-block directive as at the first turn, which contains no verdict text, and the policy emits a fresh plan; no verifier output ever enters the prompt. Within an attempt, a corrective action reaches the policy only through the environment: the policy observes the corrected state as an ordinary action–observation transition and conditions on that state rather than on advice. Across the reflective attempts of §3.4, each attempt begins with a fresh plan.

Information boundary. The verifier reads the plan, the current step, the last $H{=}5$ action–observation pairs, and the latest action. It does not see the environment state, the reward, the task solution, or the admissible-action list that the policy's prompt contains; its information is a strict subset of the policy's, and its weights are the policy's. It is a state-grounded step verifier, not an action translator: it decides whether the declared step has been realised in the observation and, if not, proposes the minimal action that would realise it. That proposal is fallible in the same way the policy is, which is why 26% of corrective actions are rejected by the environment and 29% re-navigate to an already-satisfied state (correction accounting below); an oracle with privileged access would produce neither

class. Supplying the admissible-action list to the verifier would remove the rejected class at no cost; we withhold it so that the verifier's information never exceeds the policy's, and note it as an improvement for future work.

Reach. Because the trigger requires the policy to emit the step's action, plan steps the policy never reaches are never checked. On ALFWorld the verifier observes about 36% of plan steps; the diagnostic run in §4.5 widened this to 75% without changing success, so the reach of the engine is bounded by the policy's exploration rather than by the trigger. This is the mechanism behind the statement that the residual failures are exploration failures.

Boundaries in other domains. The tracker is defined over action types, not over ALFWorld's vocabulary: each plan step begins with an action type, and the trigger matches the emitted action's type and target against it. On ALFWorld the action types are the environment verbs (*go to*, *take*, *open*, *heat*, ...); on multi-hop QA they are *search* and *answer*; on WebShop they are *search*, *click*, and *buy*. Porting the engine to a new domain therefore changes only this type table.

The `schema_ok` *field.* The verifier's JSON also carries a `schema_ok` flag recording whether the latest action was accepted by the environment's grammar. It is logged for the analyses in this appendix and is not used for control; the only fields that affect execution are `landed` and `corrective_action`.

Algorithm 1 SKT: one trial under both engines

Require: frozen policy π_θ, environment, task g, library $\mathcal{K}$, lesson buffer $\mathcal{L}$

$\mathcal{K}_{\text{top}} \leftarrow \textsc{Retriever}(g, \mathcal{K})$ ▷ top-k=3
$\text{prompt}_\theta \leftarrow \text{system}(\mathcal{K}_{\text{top}}) \oplus \text{user}(g)$ ▷ constructed once, never amended
$P \leftarrow \text{parse_plan}(\pi_\theta(\text{prompt}_\theta))$; $c \leftarrow 1$; pending $\leftarrow \bot$; obs $\leftarrow \text{env.reset}(g)$
while not terminated and $t < T$ **do**
  prev_corr $\leftarrow$ (pending $\neq \bot$)
  $a \leftarrow$ pending if pending $\neq \bot$ else $\pi_\theta(\text{obs}, \text{hist})$; pending $\leftarrow \bot$
  obs $\leftarrow \text{env.step}(a)$; hist.append$((a, \text{obs}))$
  if prev_corr **then** ▷ verify-after-fix
    $v \leftarrow J.\textsc{VerifyAfterFix}(s_c, a, \text{obs})$
    if v.now_landed **then** $c \leftarrow c + 1$
    else if v.next_action $\neq \bot$ **then** pending $\leftarrow v$.next_action
    else if v.escalate **then** flag replan; **break**
    end if
  else if a triggers boundary for s_c **then** ▷ step-realisation check
    $d \leftarrow J.\textsc{StepCheck}(P, s_c, \text{hist}, a)$
    if d.landed **then** $c \leftarrow c + 1$
    else if $d.a^c \neq \bot$ **then** pending $\leftarrow d.a^c$ ▷ executed directly next turn
    end if
  end if
end while
$y \leftarrow$ env.success
if y **then** $\mathcal{K} \leftarrow \textsc{Creator}(\tau, \mathcal{K})$
else $\mathcal{K} \leftarrow \textsc{Refiner}(\tau, \mathcal{K}_{\text{top}})$; $\mathcal{L} \leftarrow \mathcal{L} \cup \{\ell\}$
end if
if $|\mathcal{L}| \geq N_{\text{fire}}$ **then** $\mathcal{K} \leftarrow \text{Distill}(\mathcal{L}, \mathcal{K})$; $\mathcal{L} \leftarrow \emptyset$
end if
return $y, \mathcal{K}, \mathcal{L}$

Worked example. On a pick-heat-then-place trial the policy emits `Plan: 1. find tomato; 2. take tomato; 3. heat tomato in microwave 1; 4. place tomato in fridge 1.` After the action `heat tomato with microwave 1` the verb match triggers a boundary. J inspects the recent window: the microwave is closed and the latest observation reads "Nothing happens." It returns `landed=false` with corrective `open microwave 1`, followed by `heat tomato with microwave 1`. The sequence executes; the next observations show the microwave open and the tomato heated; verify-after-fix returns

`now_landed=true`; the tracker advances to step 4 and control returns to the policy, whose prompt was never modified.

Per-trial overhead. Auxiliary calls are the retriever (one per trial), step-realisation checks (one per boundary, at most $M \approx 3$–6 per trial), verify-after-fix (one per failed step), and creator-or-refiner (one per trial), an $O(M)$ overhead; the distiller amortises one call every N_{fire} failed trials.

Correction accounting. On the single held-out run of Table 10 the verifier was invoked 855 times, of which 519 concluded a boundary had been reached and 341 emitted a corrective action, spread over 94 trials; 5 of those trials succeed only because of the corrections, matching the $+3.73\,pp$ of that run, and the three fresh seeds rescue 4, 4, and 5 trials respectively. Recounted from the logged calls, the verifier consumed about 0.33M input and 0.038M output tokens per run on the frozen 7B backbone (for scale, about $0.06 if billed at a commercial API's list price) and added 12.7 minutes of wall-clock per run on average (+3.1%; per seed $+11.0$, $+28.9$, and -1.6 minutes, the negative value being shared-machine timing noise). Re-examining every corrective action on the three fresh seeds for whether its targeted condition held afterwards gives 32.7% of 898 correction records, or 24.75% of 711 corrections under a strict rule that counts a correction as ineffective whenever its repeated examinations disagree. The six-class taxonomy of all 898 records is: effective 294; redundant re-navigation (state already satisfied) 258; non-admissible command (rejected by the environment) 235; precondition unmet 87; misdirected (wrong object or place) 19; indeterminate 5. The diagnostic run referenced in §4.5 widened the share of plan steps the verifier can observe and judge from 36% to 75% (86% agreement with a manual spot-check on the added steps) and left success unchanged at 7, 6, and 5 of 50 sampled tasks per seed.

Channel control, full table. Table 15 gives the per-seed values behind Figure 4(b) and the decomposition in §4.5. The prompt-advice arm renders the verifier's `corrective_action` as a one-line suggestion appended to the policy's next user message instead of executing it; the prompt-neutral arm appends a neutral sentence of matched token length that carries no verdict; both invoke the identical verifier on the identical schedule.

Table 15: **Same-signal channel control** (ALFWorld held-out tasks, 3 seeds). The verifier, its context, its firing schedule, and the code path are identical across arms; a single flag selects what is done with the verdict. Δ is the paired per-seed increment over the arm in the row above, so the three increments sum to the total correction effect of $+3.23\,pp$; "sign" counts seeds on which the increment is positive.

Arm	What is done with the verdict	seed 41	seed 42	seed 43	mean $\pm$ std	Δ vs. row above	sign
no verifier	(verifier never called)	54.48	55.97	56.72	55.72 ± 1.14	—	—
prompt-neutral	discarded; equal-length neutral text appended to the prompt	55.97	54.48	57.46	55.97 ± 1.49	$+0.25 \pm 1.55$	2/3
prompt-advice	appended to the prompt as advice; the policy decides	55.97	57.46	56.72	56.72 ± 0.75	$+0.75 \pm 1.97$	1/3
direct execution (SKT)	executed as the next environment action	**57.46**	**58.96**	**60.45**	$\mathbf{58.96 \pm 1.50}$	$+2.24 \pm 1.29$	3/3

Submission-era comparison with published prompt-side verifiers. Before the matched three-seed campaign, we layered three published verification designs on the same library-only configuration, in the same harness, on a single seed (Table 16). Each delivers its signal through the policy prompt: REFLACT injects a goal-state reflection before every action, the VOYAGER-style critic injects a critique after each action, and the step verifier injects a "wrong, retry" signal without a corrective. All three lowered success, whereas the Critique Engine raised it. Because these designs differ from ours in the verifier prompt and the firing schedule as well as in the channel, this comparison motivates the channel control of Table 15 rather than replacing it; we report it for completeness.

Table 16: Single-seed, same-harness comparison of verification designs layered on the library-only configuration (ALFWorld held-out tasks; submission-era run).

Verification design	Channel	Success (%)	Δ vs. library only
Library only (no verification)	—	55.97	—
+ REFLACT (Kim et al., 2025)	prompt	49.25	-6.72
+ VOYAGER-style critic (Wang et al., 2023)	prompt	55.22	-0.75
+ Step verifier (Gou et al., 2024; Chen et al., 2024b)	prompt	50.75	-5.22
+ Critique Engine (ours)	direct actuation	59.70	$+3.73$

Tracker reliability. Over three full held-out runs the plan parser failed on 0 of 805 emissions (cross-checked on 134/134 trials; 100% on 14B as well). On a 300-step stratified sample labelled by one of the authors, blinded to the tracker and verifier decisions, with a second author reviewing a random 30-item subset, boundary precision is 97.99% and recall 96.05%, i.e. a false-trigger rate of 2.0% (95% CI $[0.7, 5.8]$) and a missed-boundary rate of 3.9% ($[1.8, 8.3]$). Both sit at the trigger layer the verifier gate filters: only a `landed=true` decision advances the plan, only `landed=false` emits a correction, and an unparsed decision abstains.

E VERBATIM PROMPTS

This section lists the verbatim text of the four operator prompts, the two verifier prompts, the plan-block directive, and the policy system-prompt template. Placeholders are in {curly_braces}.

```
You are an expert agent operating in an Embodied Environment.
Your task is to: {task_description}

## Retrieved Relevant Experience

{retrieved_memories}

## Current Progress

Prior to this step, you have already taken {step_count} step(s).
Below are the most recent {history_length} observations and the
corresponding actions you took: {action_history}
You are now at step {current_step} and your current observation is:
{current_observation}
Your admissible actions of the current situation are:
[{admissible_actions}].

Now it's your turn to take an action.
You should first reason step-by-step about the current situation.
This reasoning process MUST be enclosed within <think> </think> tags.
Once you've finished your reasoning, you should choose an admissible
action for current step and present it within <action> </action> tags.



### Episode Planning -- MANDATORY inside <think>
Your <think>...</think> reasoning block MUST BEGIN with a line of the
exact form:
Plan: 1. <verb-first step> 2. <verb-first step> 3. <verb-first step>

Only AFTER the Plan line, continue with any other reasoning, then
close </think> and emit <action> as usual. Each later step's
<think> should reference which plan step is being advanced.

Example:
<think>
Plan: 1. go to fridge 1 2. open fridge 1 3. take apple 1
4. go to countertop 2 5. put apple 1 in countertop 2
The apple is typically in the fridge. Step 1 is to go there.
</think>
<action>go to fridge 1</action>
```

Operator 1/4: skill-retriever. The task_type used to pre-filter candidates is parsed from the task description string (for example, "clean" or "heat" in the instruction), not read from the environment; every arm that uses a retriever (SKT, SKT w/o Critique, SKILLNET) receives the same parsed label.

```
---
name: skill-retriever
role: meta
purpose: Select the most relevant domain skills for a new ALFWorld task.
invoked_per: episode_start
expected_output_format: json_list_of_strings
---

# Skill Retriever
```

```
You are the **Skill Retriever** for an ALFWorld agent. You receive
a new task and up to 5 candidate skills (pre-filtered by task_type).
Return a JSON list of the **top-3 most useful** candidate names, in
descending order of relevance.

## Input (provided in user message)

- `task_description`: natural-language task (e.g., "heat some apple
  and put it in fridge")
- `task_type`: one of `pick_and_place`, `pick_two_obj_and_place`,
  `look_at_obj_in_light`, `pick_heat_then_place_in_recep`,
  `pick_cool_then_place_in_recep`,
  `pick_clean_then_place_in_recep`
- `candidates`: list of up to 5 skills, each with full `name` /
  `description` / `task_types` / `instructions` body (and sometimes
  `anti_patterns`).

## Output (exact format)

A JSON array of at most 3 skill **names** from the candidate set,
in descending relevance:
["clean-bowl-in-sink-and-place", "track-examined-receptacles",
 "household-object-location-priors"]

Return [] if NONE of the candidates are genuinely applicable.

## Selection rules

1. **Read the full `instructions`** of each candidate, not just
   name/description.
2. Prefer skills whose instructions reference the **specific target
   object** (e.g. "apple", "bowl") or the **specific target
   receptacle** (e.g. "fridge", "sinkbasin") from the
   task_description.
3. Prefer skills whose instructions explicitly cover the task's
   **required action sequence** (heat -> fridge; clean -> place;
   look_at -> lamp).
4. **Deprioritize anti-pattern-only skills** (names starting with
   `avoid-`, `forget-`, `dont-`): include them only if the task's
   phrasing suggests the same pitfall the skill warns against.
5. **Avoid redundancy**: if two candidates cover the same
   sub-pattern, keep only the more specific one.
6. Return ONLY the JSON array -- no prose, no backticks, no
   commentary. Invalid output is treated as empty selection.


---
name: skill-creator
role: meta
purpose: On task success, decide whether to distill a new domain
  skill from the trajectory.
invoked_per: episode_end_success
expected_output_format: json_object
---

# Skill Creator

When an agent SUCCEEDS on a task, you decide whether the
winning trajectory reveals a **novel, reusable pattern** worth
adding to the library. If so, author it as structured SKILL.md
content.

## Input (provided in user message)
```

```
- `task_description`
- `task_type`
- `trajectory`: ordered list of the agent's actions
  (`<action>...</action>` content; empty string means the model
  failed to emit an action that step)
- `final_observation`
- `reward_sum` (will be >= 10 since you are only invoked on success)
- `existing_skills_by_type`: short table (skill_name -> one-line
  description) grouped by task_type, showing what the library
  already has.

## Output (EXACT JSON, no backticks, no prose)

{
    "decision": "CREATE" | "SKIP",
    "reason": "<one sentence>",
    "skill": {
      "name": "<kebab-case-unique-name>",
      "description": "<one-line what this teaches, <= 160 chars>",
      "instructions": "1. step one.\n2. step two.\n3. step three.",
      "task_types": ["<trial task_type>"]
    }
}

If `decision == "SKIP"`, **omit** the `skill` field entirely.

## CREATE vs SKIP rules

- **SKIP** if `existing_skills_by_type` already contains a skill
  that covers the same pattern (same object class x same task_type).
  Adding a duplicate dilutes future retrieval.
- **SKIP** if the trajectory has fewer than 6 meaningful (non-empty)
  actions -- not enough signal to generalize from.
- **CREATE** if the trajectory reveals any of:
  - a **non-obvious receptacle location** for the target object
    class,
  - a **specific action ordering** the agent used to succeed,
  - a **task-type-specific workflow** (e.g., clean-before-place,
    heat-before-place) that no existing skill for this type covers.

## Skill content rules

1. `name`: unique in library, kebab-case, starts with an action.
   **CRITICAL -- NEVER include specific object instance names or
   IDs**: forbidden are `pot`, `apple`, `mug`, `bowl`, `dish`,
   `saltshaker`, `knife 1`, `cabinet 3`, etc. Use generic class
   names ONLY: `object`, `item`, `container`, `surface`, `target`,
   `receptacle`. GOOD: `heat-object-and-place-in-receptacle`. BAD:
   `heat-pot-in-microwave` (uses "pot").
2. `description`: one sentence, <= 160 chars. Describes WHAT the
   skill teaches abstractly. Forbidden concrete object/instance
   references (same list as rule 1). Use "the target object", "the
   item", "a container".
3. `instructions`: numbered steps (`1.`, `2.`, `3.`, ...) in the
   same order the successful trajectory used. **Use generic class
   names throughout**: "cabinet" not "cabinet 1", "drawer" not
   "drawer 3", "the target object" not "the apple" or "the knife".
   The pattern must transfer across object instances.
4. `task_types`: MUST include the trial's `task_type`. Add
   `"general"` ONLY if the pattern is truly type-agnostic (e.g.,
   "always open drawers before examining their contents"). Prefer
   single-type tags over general.
5. NO anti-pattern content here -- the refiner owns that.
6. If you cannot write a generic enough skill (i.e., the trajectory
   insight is truly object-specific), output `"decision": "SKIP"`
```

```
   instead of creating an over-fitted skill.



---
name: skill-refiner
role: meta
purpose: On task failure, patch the most-relevant retrieved skill
  with anti-patterns or extended instructions.
invoked_per: episode_end_failure
expected_output_format: json_object
---

# Skill Refiner

When an agent FAILS a task, you diagnose which retrieved
skill was most relevant to the failure and decide whether to
**patch** its instructions / add an anti-pattern section, or
**no-patch** if the failure is beyond the skill's scope.

## Input (provided in user message)

- `task_description`, `task_type`
- `trajectory`: agent's action list (empty string means model failed
  to emit a valid action that step)
- `final_observation`
- `reward_sum` (= 0 since failure)
- `failure_reason`: e.g. `max_steps_reached(50)` or
  `early_termination_step=12_reward_sum=0.00`
- `retrieved_skills`: the up-to-3 skills that were injected into the
  agent's prompt for this trial. Each has `name`, `description`,
  `task_types`, `instructions`, and (optionally) existing
  `anti_patterns`.

## Output (EXACT JSON, no backticks)

{
    "decision": "PATCH" | "NO_PATCH",
    "reason": "<one-sentence diagnosis>",
    "target_skill_name": "<exact name from retrieved_skills>",
    "patched_skill": {
      "name": "<same as target_skill_name>",
      "description": "<unchanged or slightly refined>",
      "instructions": "<full updated multi-line numbered steps>",
      "task_types": [<unchanged from original>],
      "anti_patterns": "- DO NOT X because Y.\n- DO NOT Z because W."
    }
}

If `decision == "NO_PATCH"`, **omit** `target_skill_name` and
`patched_skill`.

## PATCH vs NO_PATCH rules

- **NO_PATCH** if failure is a **model-capability issue**:
  trajectory shows > 50% empty `<action>` entries, the agent never
  took any task-relevant action, or reasoning was incoherent. No
  skill text can fix that.
- **NO_PATCH** if the 3 `retrieved_skills` were **irrelevant** to
  the failure mode (agent failed in a phase none of the skills
  addressed).
- **PATCH** if trajectory shows the agent **followed** a retrieved
  skill's step but it led astray (wrong receptacle priority,
  missing prerequisite) -> add a specific anti-pattern warning +
```

```
  extend instructions.
- **PATCH** if the retrieved skill was **close but incomplete**
  (e.g., missing a step the successful path requires) -> extend
  instructions only.

## Patched content rules

1. `target_skill_name` MUST exactly match one of the input
   `retrieved_skills` names.
2. `anti_patterns` is a markdown bullet list in the form
   `- DO NOT X because Y.` If the skill already had `anti_patterns`,
   APPEND new bullets -- do not replace existing ones.
3. **CRITICAL -- patched content uses GENERIC class names**: in
   both `instructions` and `anti_patterns`, do NOT name specific
   object instances like "pot", "apple", "knife 1", "cabinet 3".
   Use "the target object", "the item", "a container", "a
   receptacle". The patched pattern must transfer across object
   instances.
4. `instructions` may be EXTENDED or clarified but NOT reordered or
   shortened from the original (the original is version-controlled
   under `_versions/`).
5. `description` and `task_types` are typically unchanged.
6. Return ONLY the JSON. Parse errors fall back to `NO_PATCH`.



You are the Skill Distiller. Given a batch of failure lessons and a
snapshot of the current skill library, you CLUSTER lessons by the
underlying reasoning breakdown they reveal and decide, per cluster,
whether to REFINE an existing skill (when the cluster overlaps one)
or CREATE a new skill. You output strict JSON only.

## task
Group the following failure lessons into coherent clusters and, per
cluster, decide whether to REFINE an existing library skill or
CREATE a new one.

{axes_block}

## existing library snapshot (for overlap detection -- prefer
##   REFINE over redundant CREATE)

{library_snapshot}

## failure lessons (n={n_lessons})
{lesson_dump}

## output

Return EXACTLY one JSON array (no backticks, no prose). Each element
is ONE cluster:

[
    {
      "axis": "<one of: Information, Planning, Execution, StateTrack,
               Goal, Termination>",
      "sub_description": "<free-form Level-2 description, <= 15 words,
                          specific to this cluster>",
      "members": ["<lesson_id>", ...],
      "prevalence": <int = len(members)>,
      "overlaps_with": "<existing skill_id or null if no good overlap>",
      "decision": "<REFINE if overlaps_with is non-null, else CREATE>",
      "skill_md": "<full SKILL.md body, YAML frontmatter + markdown
                   -- see format below>"
```

```
    },
    ...
]

### skill_md format

---
name: <kebab-case-name>
description: <one line, <= 160 chars>
task_types: ["<opt1>", ...]   # may be [] if benchmark has no task_type
axis: <one of the 6 axes>
sub_description: <same as cluster.sub_description>
---

# <name>

<one-paragraph description>

## Instructions
1. step one
2. step two
...

## Anti-patterns
- <failure mode 1>
- <failure mode 2>

### length constraints (strict)
- Keep each `skill_md` body <= 250 tokens.
- Instructions section: <= 5 numbered lines.
- Anti-patterns section: <= 3 bullets.

### rules
1. **No object instance IDs or instance names** anywhere in
   `skill_md` (forbidden: `knife 1`, `cabinet 3`, etc.). Use
   generic class nouns: `object`, `item`, `container`,
   `receptacle`, `surface`.
2. NEVER use these specific object names in `skill_md`: apple, pot,
   mug, egg, dish, plate, tomato, bread, potato, spatula, cup, bowl,
   pan, saltshaker, peppershaker, lettuce, butterknife, cellphone,
   cloth, soapbar, candle. Describe via generic class.
3. Prefer REFINE when an existing skill already covers the same
   pattern -- set `overlaps_with` to that skill_id and `decision`
   to `REFINE`. When REFINE, `skill_md` should restate the
   existing skill's `name` with improved instructions /
   anti-patterns.
4. CREATE only when no existing skill covers the cluster -- set
   `overlaps_with` to `null` and `decision` to `CREATE`.
5. A cluster with fewer than 3 members will be dropped downstream;
   do not try to inflate counts, but also do not split a coherent
   group unnecessarily.
6. Every `lesson_id` you list in `members` must appear in the input
   (no hallucinated IDs). Each lesson may appear in at most one
   cluster.
7. Output JSON ONLY. No backticks, no commentary.



You are an external judge for an ALFWorld agent.

Task: {task}

Agent's plan (numbered steps):
{plan}
```

```
The agent is currently expected to be working on step {step_idx}
({step_idx_oneindex} of {n_steps}):
  "{step_text}"

Recent (action, observation) pairs (last {n_history}):
{history}

Your job: judge whether plan step {step_idx_oneindex}
("{step_text}") has just LANDED based on the LATEST post-action
observation (the line tagged "LATEST post-action obs" in the history
block above -- NOT older history entries). The "evidence_obs" field
in your output MUST be a direct excerpt from that LATEST obs, not
from earlier history. Also judge whether the agent's last action
was schema-valid (ALFWorld accepts it, did not respond with
"Nothing happens." or grammar-error).

Output JSON ONLY (no markdown fences, no surrounding text):
{
  "landed": true|false,
  "schema_ok": true|false,
  "reason": "<one-sentence justification>",
  "evidence_obs": "<excerpt from latest obs that proves landed/not>",
  "corrective_action": "<concrete ALFWorld action OR null>"
}

Rules:
  - "landed" = true ONLY if the latest observation contains evidence
    that the step's intended effect was achieved (e.g., step
    "open fridge 1" landed iff obs mentions fridge 1 is open and
    lists its contents).
  - "schema_ok" = false if the latest observation is literally
    "Nothing happens." or contains "you can't" / "is closed".
    Else true.
  - "corrective_action" = a single concrete ALFWorld action that
    should be tried next to make this step land. Only set when
    landed=false. Use ALFWorld grammar (e.g., "open fridge 1",
    "go to countertop 2", "take apple 1 from fridge 1"). Set to
    null if landed=true OR if you cannot suggest a specific
    corrective action.

Output the JSON only.
```

```
You are an external judge for an ALFWorld agent.

Plan step being attempted: "{step_text}" (step {step_idx_oneindex}
of plan).

Previous corrective action you suggested: "{prev_corrective}"

Observation AFTER applying that corrective action:
{actual_obs}

Your job: judge whether the corrective action actually worked, i.e.,
whether the plan step has now landed. If not, decide: (a) propose
another single concrete next_action OR (b) escalate to plan-level
replan.

Output JSON ONLY (no markdown fences, no surrounding text):
{
  "fix_worked": true|false,
  "now_landed": true|false,
```

```
  "next_action": "<concrete ALFWorld action OR null>",
  "should_escalate_replan": true|false
}

Rules:
  - "fix_worked" = true if the corrective action visibly changed env
    state in the intended direction (open succeeded, navigation
    arrived, take succeeded, etc.).
  - "now_landed" = true if the original plan step is now achieved
    (often == fix_worked for single-action steps, but may differ
    for multi-step subgoals).
  - "next_action" = a single concrete next action to try. Only when
    fix_worked=false AND should_escalate_replan=false. Else null.
  - "should_escalate_replan" = true ONLY when the observation
    reveals the original plan is fundamentally wrong (target object
    is in a different room/receptacle than the plan assumed).
    Else false.

Output the JSON only.
```

F CASE STUDIES

Each case is drawn from the held-out tasks on which SKT succeeds while the three inference-time baselines fail.

pick_cool_then_place: "cool some lettuce and put it in countertop." Two pitfalls drive the baseline failures: the lettuce sits on countertop 2 rather than the procedure-default countertop 1, and the goal countertop differs from the pickup surface. REFLEXION stalls after the cool step and loops *go* actions until the budget runs out; EXPEL's insights bias the search toward storage receptacles and never visit countertop 2; SKILLNET's hand-authored locate phase exhausts the budget on cabinets and drawers. SKT retrieves `cool-object-and-place-in-countertop` (Creator-origin) and `cool-object-and-examine-before-placing` (Distiller-origin), both of which encode "examine candidate countertops before *put*", and succeeds in 13 actions. The Critique Engine never fires on this trial, so the win is attributable to library composition alone.

pick_clean_then_place: "put a clean knife in countertop." The knife is on countertop 2 and the goal is countertop 1; ALFWorld rejects *put* on the source countertop. REFLEXION tries `put knife on countertop 1, 2, 3` sequentially, each returning "Nothing happens", and exhausts its budget in a reflection loop; EXPEL's insights overweight drawer storage and it never reaches the countertops; SKILLNET's place-phase template hard-codes countertop 1 with no recovery branch. SKT retrieves `clean-object-and-place-in-receptacle`, `examine-sink-before-cleaning-and-placing`, and `examine-drawers-and-countertops` (all Creator-origin), which enumerate candidate countertops before *put*, and succeeds on the first attempt in 18 actions with no verifier call.

pick_and_place: "put some mug on desk" (bedroom layout). The mug is on shelf 1, not in a kitchen receptacle, and the goal is desk 1. All three baselines search drawers with a kitchen prior and never visit the shelf. SKT retrieves `go-to-shelf-then-toilet`, written from a collection-phase trial in another bedroom layout, which suggests shelf 1 as the first lookup; the trial completes in 4 actions with zero failed actions.

pick_two_obj_and_place: "find two soapbar and put them in cabinet" (multi-attempt case). The modal failure is instance collapse: the policy places the first soapbar, returns, and re-takes the same one, or attempts to take the second while its inventory is full. SKT retrieves three Distiller-origin skills (`lost-track-of-progress`, `incorrect-task-decomposition`, `early-termination`) that encode the correct [pick$_1$, place$_1$, pick$_2$, place$_2$] order. The first two attempts still fail at the budget; on the third, the cumulative reflection and the failure-mode skills together tip the policy toward the corrected sequence and it succeeds. This is a combined Skill-Engine and reflective-retry win.