SkillBrew: Multi-Objective Curation of Skill Banks for LLM Agents

Wentao Hu[♠*], **Zhendong Chu**[♡*†], **Yiming Zhang**[♣], **Junda Wu**[♢], **Ming Jin**[★], **Xiangyu Zhao**[♠], **Yilei Shao**[△], **Yanfeng Wang**[□], **Qingsong Wen**[♡†]
[♠]City University of Hong Kong [♡]Squirrel Ai Learning
[♣]University of Science and Technology of China [♢]University of California, San Diego
[★]Griffith University [△]East China Normal University [□]Shanghai Jiao Tong University
zc9uy@virginia.edu, qingsongedu@gmail.com

Abstract

Retrieval-augmented LLM agents increasingly rely on curated skill banks: collections of reusable textual principles that guide decision making on complex tasks. Existing approaches typically expand these banks in an append-only fashion, continuously adding new skills without removing redundant, outdated, or harmful ones, resulting in inefficient and poorly curated repositories. In this paper, we formulate skill bank curation as a constrained multi-objective problem: a desirable bank must be *useful* for the agent, *diverse* in its content, and provide good *coverage* of the query distribution. To this end, we introduce **SkillBrew**, a multi-objective curation framework that formalizes skill bank curation as Pareto-aware optimization under a utility constraint, and solves it via a bi-level propose-then-verify loop. We evaluate our approach on three public benchmarks. Our findings suggest that treating skill banks as objects of principled curation, rather than ever-growing append-only logs, is an important step toward building self-improving LLM agents. Code is available at https://github.com/Applied-Machine-Learning-Lab/EMNLP26_SkillBrew.

1 Introduction

Large language model (LLM) agents (Yao et al., 2022b; Shinn et al., 2023) are increasingly deployed on long-horizon, open-ended tasks where the ability to accumulate and reuse procedural knowledge across episodes determines downstream performance. A widely adopted abstraction is the *skill bank*: a retrievable collection of distilled, reusable textual or executable artifacts that encode how the agent performs recurring tasks (Wang et al., 2023; Zhao et al., 2024; Ni et al., 2026). This abstraction has been standardized by Anthropic's Agent Skills specification (Anthropic, 2025) and is now adopted by modern agent harnesses such as Claude Code. However, the skill bank is not a one-shot artifact: a bank that is simultaneously useful, diverse, and well-covered cannot be obtained by a single distillation pass, and requires iterative refinement that balances these competing properties (Wang et al., 2023; Fang et al., 2026). Yet how to build, refine, and prune skills jointly as a bank, rather than judging each skill in isolation, remains largely unexamined.

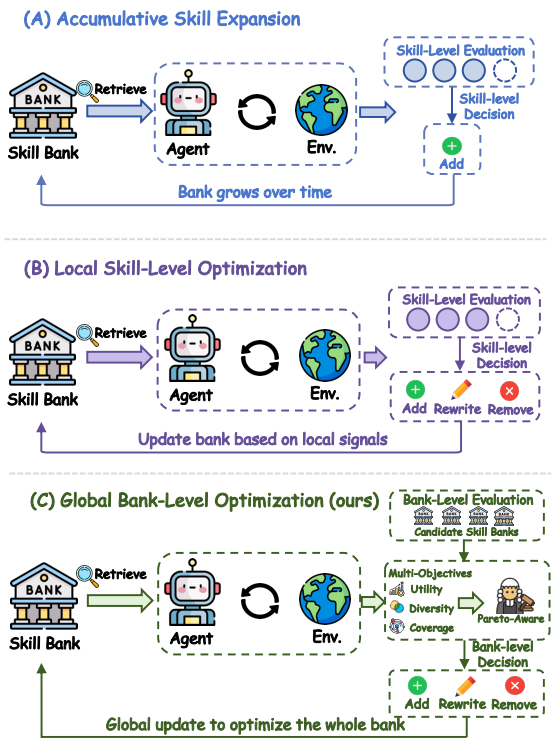

Figure 1: Three paradigms of skill bank curation. Prior work either (A) accumulates skills without removal, or (B) edits skills based on per-skill signals. We propose (C) global bank-level optimization, which evaluates the entire bank under multi-objective trade-offs and chooses updates via a Pareto-aware selector.

A growing line of work tackles skill bank con-

[*]Equal contribution. Work done during WH's internship at Squirrel Ai Learning, under the mentorship of ZC.
[†]Corresponding author.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 14373–14391
October 24-29, 2026 ©2026 Association for Computational Linguistics

struction, which we organize into two paradigms (Figure 1). The first, *accumulative skill expansion*, continuously adds new skills from experience and lets the bank grow over time, as in Voyager's self-verified skill library (Wang et al., 2023), Trace2Skill's hierarchical consolidation of trajectory-local lessons (Ni et al., 2026), and SkillRL's failure-driven skill evolution coupled with RL policy updates (Xia et al., 2026). The second, *local skill-level optimization*, retains, edits, or removes skills based on a single scalar curation signal, such as ExpeL's vote-based ADD/EDIT operations on cross-task insights (Zhao et al., 2024), top-k frontier selection on held-out accuracy (Alzubi et al., 2026), and non-parametric scoring gates (Mi et al., 2026). While these methods advance skill bank construction along different axes, their curation reduces to a single scalar criterion, and bank-level properties such as diversity and coverage remain a less explored dimension.

Principled skill bank curation is challenging for three interrelated reasons. First, the quality of a bank is inherently *multi-dimensional*: a good bank must be useful, internally diverse, and broadly cover the query distribution. These criteria are mutually constraining: adding skills to broaden coverage tends to introduce redundancy that lowers diversity, while pruning to preserve diversity reduces coverage; they cannot be collapsed into a single scalar without introducing pathologies. Second, *credit assignment* is difficult: the contribution of any individual skill is observable only through expensive rollouts, and is further confounded by interactions with co-retrieved skills, making it hard to attribute outcomes to specific entries. Third, the objectives are *non-differentiable and expensive to query*, while the space of admissible edits is open-ended, since new skills are synthesized from trajectory evidence rather than chosen from a fixed pool. This rules out gradient-based optimization and exhaustive search, so curation requires a mechanism that can both propose plausible edits and verify them under the multi-objective criterion. These challenges explain why prior work has defaulted to accumulation without pruning or single-scalar curation rather than principled multi-objective curation.

In this paper, we propose **SkillBrew**, a training-free framework for iterative skill bank refinement that introduces a third paradigm, *global bank-level optimization* (Figure 1C), and addresses the above challenges through three design choices. First, we formalize skill bank curation as *constrained multi-objective optimization* over utility, diversity, and coverage, with utility as an explicit constraint and diversity/coverage as structural regularizers, reflecting the asymmetry that a bank must first be useful before its organization matters. Second, we attribute credit through *per-skill counterfactual replay*: leave-one-out rollouts isolate the marginal contribution of each skill, providing fine-grained KEEP/REWRITE/REMOVE evidence for bank-level optimization. Third, we navigate the non-differentiable and open-ended edit space via a *bi-level propose-then-verify loop*, which uses trajectory evidence on a support split to propose candidate edits and Pareto-aware selection on a held-out query split to verify them, preventing overfitting to the trajectories that motivated each edit.

Across three public benchmarks, **SkillBrew** substantially outperforms strong baselines and transfers across worker LLMs of different scales, suggesting that treating skill banks as objects of principled curation, rather than judging skills in isolation, yields more effective procedural knowledge for retrieval-augmented agents.

2 Related Work

2.1 From Agent Memory to Agent Skill

A central challenge in deploying LLM agents (Yao et al., 2022b) on long-horizon tasks is that each task is typically treated as an isolated episode, leaving no straightforward mechanism to accumulate transferable knowledge across tasks. One line of work equips the agent with a persistent memory of past interaction traces (Chhikara et al., 2025; Fang et al., 2026), retrieved at inference time as references for similar future tasks. ReasoningBank (Ouyang et al., 2026) pushes this direction further by using an LLM extractor to distill both self-judged successful and failed trajectories into structured reasoning memory, yet it adopts a minimal consolidation strategy in which new items are appended without pruning and are selected purely by retrieval relevance.

A distinct line abstracts raw experience into skills: compact and reusable procedural artifacts that capture how agents perform recurring tasks. Voyager (Wang et al., 2023) first brought this paradigm to LLM agents by maintaining a growing library of executable code skills in Minecraft, and subsequent work generalizes the formulation to textual forms, including cross-task insights dis-

tilled from trajectories (Zhao et al., 2024) and transferable skills consolidated from trajectory pools via parallel hierarchical analysis (Ni et al., 2026). This paradigm has further been standardized by Anthropic's Agent Skills specification (Anthropic, 2025). Across these designs, the skill bank serves as a compact retrievable representation of procedural knowledge, but how to curate this bank iteratively, balancing the utility of individual skills against the diversity and coverage of the bank as a whole, remains largely open.

2.2 Skill Bank Curation

Early work on skill libraries, exemplified by Voyager (Wang et al., 2023), treats the bank as an append-only artifact that grows as the agent succeeds, without mechanisms to refine or prune stored skills. More recent work introduces explicit curation over individual skills. ExpeL (Zhao et al., 2024) updates cross-task insights through vote-based ADD/EDIT operations, CoEvoSkills (Zhang et al., 2026) uses a surrogate verifier for failure feedback, Skill-Pro (Mi et al., 2026) admits and prunes skills through online scoring, and Skill-Forge (Liu et al., 2026b) closes a failure-driven refinement loop. Memento-Skills (Zhou et al., 2026) maintains a skill library that only grows with no operation that removes a learned skill, scores each skill by a single scalar and further trains a dedicated retrieval router rather than remaining training-free. However, the curation criterion in these methods collapses to a single scalar signal, folding utility, diversity, and coverage of the bank into one ordering, so the trade-offs that arise during curation cannot in principle be resolved.

3 Methodology

In this section, we present **SkillBrew**, a training-free framework for principled skill bank curation in LLM agents, as illustrated in Figure 2. We first introduce the core principles underlying effective skill bank construction, formalizing the desiderata of a curated bank in terms of *utility*, *diversity*, and *coverage*. We then cast skill bank curation as a constrained multi-objective optimization problem, where utility serves as the primary constraint while diversity and coverage regularize the bank structure. To solve this problem, we develop a bi-level propose-then-verify loop that iteratively generates candidate edits from trajectory evidence and selects non-dominated banks under a utility constraint.

3.1 Problem Definition

Let $\mathcal{S}$ denote the space of admissible skills, where each skill $s \in \mathcal{S}$ is a piece of reusable procedural knowledge that can be retrieved to guide agent behavior. A skill bank is defined as a subset $\mathcal{B} \subseteq \mathcal{S}$ serving as the retrieval pool for a frozen agent π_{agent}.

Given a task query t, a retriever RET returns a subset of relevant skills $R_t = \text{RET}(\mathcal{B}, t) \subseteq \mathcal{B}$, and the agent rolls out a trajectory

$$\tau_t = \pi_{\text{agent}}(t \mid R_t) \tag{1}$$

conditioned on R_t.

A desirable skill bank should satisfy three complementary principles. First, retrieved skills should be *useful*, meaning they improve the agent's task performance. Second, the bank should be *diverse*, avoiding redundant or overly similar skills. Third, the bank should provide broad *coverage* over the query distribution so that useful skills can be retrieved across a wide range of tasks. These principles are in tension: a bank of mutually orthogonal skills attains high diversity but offers little utility if the skills are off-task; aggressively pruning to a few high-utility skills sacrifices coverage on the long tail. We therefore cast skill bank curation as a **constrained multi-objective optimization**:

$$\begin{aligned} \mathcal{B}^\star \in \operatorname*{Pareto\text{-}arg\,max}_{\mathcal{B}\in\mathcal{F}} & \\ \Big(J_{\text{util}}(\mathcal{B}), J_{\text{div}}(\mathcal{B}), J_{\text{cov}}(\mathcal{B})\Big), & \\ \mathcal{F} = \{\mathcal{B} \subseteq \mathcal{S} \mid J_{\text{util}}(\mathcal{B}) \geq \eta\}\,. & \end{aligned} \tag{2}$$

where J_{util}, J_{div}, and J_{cov} respectively measure utility, diversity, and coverage, and η is a minimum utility threshold. The solution $\mathcal{B}^\star$ is Pareto-optimal (Censor, 1977): no feasible bank improves one objective without sacrificing another.

Importantly, we treat utility as the primary constraint, while diversity and coverage act as structural regularizers over the bank. This reflects the asymmetry of the three principles: a skill bank must first be useful to the agent, while diversity and coverage improve its organization, robustness, and generalization. In the next section, we describe the computation of the three objectives in detail.

We partition the task distribution into a support set $\mathcal{D}_{\text{support}}$ and a query set $\mathcal{D}_{\text{query}}$. In this paper, all three objectives $\Big(J_{\text{util}}, J_{\text{div}}, J_{\text{cov}}\Big)$ are computed from agent rollouts on $\mathcal{D}_{\text{query}}$, while $\mathcal{D}_{\text{support}}$ is reserved for proposing candidate edits in Section 3.3.

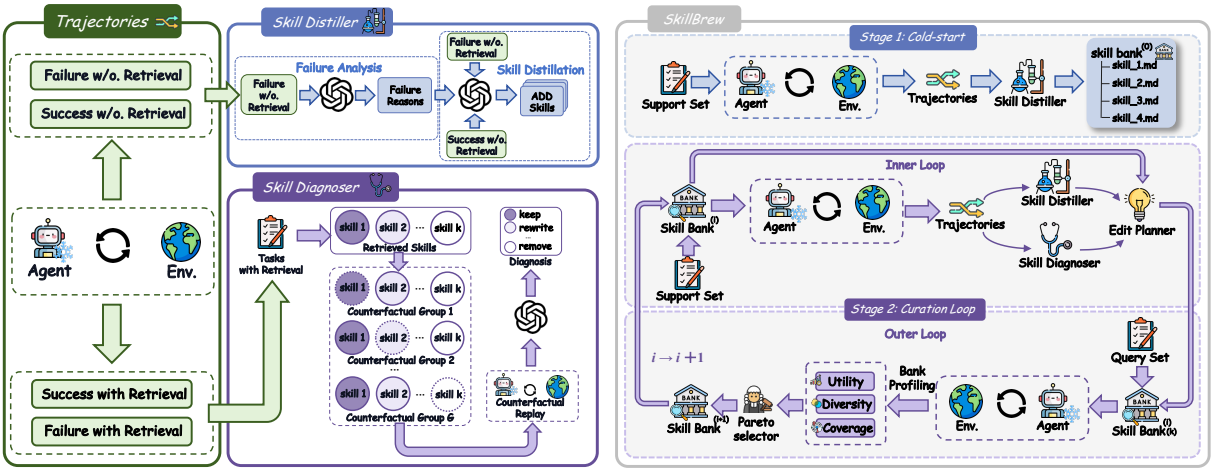

Figure 2: Overview of SkillBrew. At each round, an inner loop produces ADD candidates via the *Skill Distiller*, KEEP, REWRITE, and REMOVE verdicts via the *Skill Diagnoser*, and the *Edit Planner* sees current skill bank and proposes K new candidate banks from these edits. An outer loop then verifies these candidate banks on the query set under three objectives J_{util}, J_{div}, and J_{cov}, and selects the next bank with a Pareto-aware selector.

This separation prevents the bank from overfitting to the trajectories that motivated its edits.

3.2 Principles of Skill Bank Curation

We consider three principles that capture complementary and potentially conflicting aspects of skill bank quality: *utility* measures whether retrieved skills improve agent performance, *diversity* measures redundancy within the bank, and *coverage* measures how broadly the bank's skills are retrieved and exercised across tasks.

Utility. We measure utility through per-skill counterfactual contribution. For a trajectory $\tau \in \mathcal{D}_{\text{query}}$ retrieving skill s, we replay the agent with s removed under the same environment, yielding

$$\Delta(\tau, s) = r(\tau, s) - r(\tau, \emptyset), \tag{3}$$

where $r(\tau, \cdot) \in [0, 1]$ is the task-level reward (success indicator or normalized score) obtained under the specified retrieval. The utility of s is computed by averaging over all trajectories retrieving it,

$$J_{\text{util}}(s) = \frac{1}{|\mathcal{T}_s|} \sum_{\tau \in \mathcal{T}_s} \Delta(\tau, s), \tag{4}$$

where $\mathcal{T}_s = \{\tau_t : s \in R_t\}$. We then aggregate skill utilities into a bank-level objective,

$$J_{\text{util}}(\mathcal{B}) = \sum_{s \in \mathcal{B}} w_s \cdot J_{\text{util}}(s), \tag{5}$$

where $w_s = |\mathcal{T}_s|/N_R$ and N_R is the number of trajectories with non-empty retrieval.

Diversity. We measure diversity using the geometric volume spanned by skill embeddings. Let G denote the Gram matrix of L2-normalized skill embeddings in $\mathcal{B}$. This objective is defined as,

$$J_{\text{div}}(\mathcal{B}) = \det(G + \lambda I)^{1/|\mathcal{B}|} \in (0, 1], \tag{6}$$

where λI is a small diagonal regularizer for numerical stability. This formulation penalizes redundant or highly correlated skills, assigning higher scores to banks with structurally diverse skills.

Coverage. We measure coverage as the product of retrieval density and skill usage:

$$J_{\text{cov}}(\mathcal{B}) = \text{density} \cdot \text{usage} \in [0, 1]. \tag{7}$$

The density term measures the average fraction of retrieval slots filled across tasks in $\mathcal{D}_{\text{query}}$:

$$\text{density} = \frac{1}{|\mathcal{D}_{\text{query}}|} \sum_{t \in \mathcal{D}_{\text{query}}} \frac{|\text{RET}(\mathcal{B}, t)|}{k_{\text{top}}}. \tag{8}$$

The usage term measures the fraction of skills retrieved at least once:

$$\text{usage} = \frac{|\{s \in \mathcal{B} : s \text{ retrieved at least once}\}|}{|\mathcal{B}|}. \tag{9}$$

Together, the two terms encourage banks that not only retrieve skills frequently across tasks, but also utilize a broad portion of the bank rather than relying on a small subset of dominant skills.

3.3 Pareto-Aware Skill Bank Curation

Directly solving Eq. 2 is intractable for two reasons: $\mathcal{S}$ itself is open-ended, with new skills synthesized from trajectory evidence rather than drawn from a fixed pool; and all objectives are only observable through expensive agent rollouts. We therefore propose a bi-level propose-then-verify framework that iteratively refines the skill bank through candidate generation and Pareto-aware selection.

Bi-level propose-then-verify loop. As shown in Figure 2, at iteration i, given the current bank $\mathcal{B}^{(i)}$, the framework alternates between an inner loop that proposes candidate edits and an outer loop that verifies candidate banks:

$$\underbrace{\mathcal{K}^{(i)} = \mathrm{Plan}\big(\texttt{Distill}, \texttt{Diagnose}; \mathcal{B}^{(i)}, \mathcal{D}_{\text{support}}\big)}_{\textbf{inner loop: candidate generation}}$$
$$\underbrace{\mathcal{B}^{(i+1)} = \mathrm{Select}\big(\mathcal{K}^{(i)}; \eta, \mathcal{D}_{\text{query}}\big)}_{\textbf{outer loop: Pareto-aware selection}}. \tag{10}$$

The support/query split introduced in Section 3.1 aligns naturally with this bi-level structure: $\mathcal{D}_{\text{support}}$ supplies trajectory evidence to the inner loop, while $\mathcal{D}_{\text{query}}$ is reserved for the outer loop's Pareto-aware selection.

Cold-start initialization. As illustrated in the cold-start stage of Figure 2, we first construct an initial bank $\mathcal{B}^{(0)}$ from trajectories collected without retrieval. Following (Zhao et al., 2024), successful and failed trajectories are passed to a *Skill Distiller*, which synthesizes an initial set of ADD-type skills from failure patterns in conjunction with successful trajectories.

Inner loop: candidate generation. As illustrated in the inner loop of Figure 2 and summarized in Algorithm 1, at each iteration the inner loop transforms $\mathcal{B}^{(i)}$ into K candidate banks through three skill curation operations: ADD, REWRITE, and REMOVE.

Rolling out π_{agent} on $\mathcal{D}_{\text{support}}$ with retrieval from $\mathcal{B}^{(i)}$ yields four trajectory quadrants defined by outcome (success/failure) and retrieval (empty/non-empty):

$$\begin{aligned}
\mathcal{T}^{+}_{\bar{R}} &= \{\tau_t : \mathrm{succ}(\tau_t),\ R_t = \emptyset\},\\
\mathcal{T}^{-}_{\bar{R}} &= \{\tau_t : \mathrm{fail}(\tau_t),\ R_t = \emptyset\},\\
\mathcal{T}^{+}_{R} &= \{\tau_t : \mathrm{succ}(\tau_t),\ R_t \neq \emptyset\},\\
\mathcal{T}^{-}_{R} &= \{\tau_t : \mathrm{fail}(\tau_t),\ R_t \neq \emptyset\}.
\end{aligned} \tag{11}$$

The no-retrieval quadrants $\mathcal{T}^{\pm}_{\bar{R}}$ expose missing capabilities in the bank and are consumed by the *Skill Distiller*, which synthesizes candidate ADD skills by clustering failure patterns in $\mathcal{T}^{-}_{\bar{R}}$ and using $\mathcal{T}^{+}_{\bar{R}}$ as positive references.

The retrieval quadrants $\mathcal{T}^{\pm}_{R}$ provide counterfactual evidence on existing skills and are consumed by a *Skill Diagnoser*. For each retrieved skill $s \in R_t$, we perform a leave-one-out replay

$$\tilde{\tau}_{t\setminus s} = \pi_{\text{agent}}(t \mid R_t \setminus \{s\}), \tag{12}$$

which isolates the marginal contribution of s. Based on the factual outcome of τ_t and the counterfactual outcome of $\tilde{\tau}_{t\setminus s}$ aggregated over all trajectories retrieving s, the *Skill Diagnoser* assigns one of three verdicts: KEEP, REWRITE, or REMOVE.

Finally, an *Edit Planner* composes the three edit pools $(\mathcal{E}_{\text{add}}, \mathcal{E}_{\text{rewrite}}, \mathcal{E}_{\text{remove}})$ into K candidate banks $\mathcal{K}^{(i)} = \{\mathcal{B}^{(i+1)}_k\}_{k=1}^{K}$, while preserving the KEEP skills as protected slots.

The full prompts used to implement *Skill Distiller*, *Skill Diagnoser*, and *Edit Planner* are listed in Appendix C.

Outer loop: Pareto-aware selection. Given the candidate set $\mathcal{K}^{(i)}$, the outer loop evaluates each candidate by its objective profile

$$\Phi(\mathcal{B}) = \big(J_{\text{util}}(\mathcal{B}), J_{\text{div}}(\mathcal{B}), J_{\text{cov}}(\mathcal{B})\big). \tag{13}$$

We also include a *null candidate*, which represents keeping the current bank unchanged. This allows the selector to make no edit when none of the proposed candidates improves the bank. Over $\mathcal{K}^{(i)} \cup \{\text{null}\}$, we first compute the non-dominated Pareto front (Deb et al., 2002) under the three objectives, then resolve the front lexicographically with J_{util} as the highest priority.

Since the absolute utility threshold η in Eq. 2 cannot be specified a priori over an open-ended skill space, we replace it with a *front-adaptive* threshold $u_{\max} - \epsilon$. Let $u_{\max}$ denote the largest J_{util} on the front, and let

$$\mathrm{Tied} = \{\mathcal{B} \in \text{front} : J_{\text{util}}(\mathcal{B}) \geq u_{\max} - \epsilon\} \tag{14}$$

denote the candidates within tolerance ϵ of the lex-optimal utility. If $|\mathrm{Tied}| = 1$, the unique tied candidate wins; otherwise we break the tie inside Tied by hypervolume contribution over the two regularizers J_{div} and J_{cov}. This yields a per-round ϵ-utility safeguard, $J_{\text{util}}(\mathcal{B}^{(i+1)}) \geq J_{\text{util}}(\mathcal{B}^{(i)}) - \epsilon$, by construction. Here ϵ defines an admissible tolerance

within which diversity and coverage may be traded against utility, rather than a degradation budget that is consumed at every iteration. The tie pool always contains the null candidate, so a round gives up utility only when doing so improves the two regularizers. The corresponding formal worst case cumulative bound after T rounds is $T\epsilon$. In practice this slack is rarely used, and we report the observed per-round utility gaps in Appendix A.4.

4 Experiments

4.1 Experimental Setup

Environments. We evaluate on three agent benchmarks. ALFWorld (Shridhar et al., 2020) is a textualized embodied benchmark spanning six household task families that require compositional action sequences. We report per-task and overall success rate (%) on its test split. WebShop (Yao et al., 2022a) is a simulated online shopping benchmark requiring multi-step navigation, option customization, and purchase decisions. ScienceWorld (Wang et al., 2022) is a text-based interactive science environment that requires the agent to perform multi-step experiments and reason over physical-world dynamics. For WebShop, we report two metrics on its test split. Score is the average normalized reward measuring attribute matching between the purchased and target product, and Succ. is the fraction of episodes where every required attribute is matched. For ScienceWorld, we report the analogous pair on its test split. Score is the average dense progress score over the required experimental procedure, and Succ. is the fraction of episodes where the procedure is completed in full.

Baselines. We compare SkillBrew against ten baselines in Table 1, all using Qwen2.5-7B-Instruct as the frozen agent worker: (1) vanilla baselines (Zero-Shot and ReAct (Yao et al., 2022b)); (2) memory-based agents (Reflexion (Shinn et al., 2023), Mem0 (Chhikara et al., 2025), MemP (Fang et al., 2026), and SimpleMem (Liu et al., 2026a)); and (3) skill-based methods (Voyager (Wang et al., 2023), ExpeL (Zhao et al., 2024), EvoSkill (Alzubi et al., 2026), and Skill-Pro (Mi et al., 2026)). These training-free baselines are evaluated under frozen-worker setting. We do not compare with training-based methods such as SkillRL, which jointly updates the worker policy and the bank, since this changes the underlying agent and is not applicable to closed-source workers. We leave joint agent-bank optimization to future work.

Implementation details. We use GPT-5.4 (OpenAI, 2026) for the *Skill Distiller*, *Skill Diagnoser*, and *Edit Planner*. For ALFWorld, we sample 200 `train` episodes as $\mathcal{D}_{\text{support}}$, and use the full `valid_seen` split as $\mathcal{D}_{\text{query}}$, and report results on the full `valid_unseen` split as the test set. For WebShop we follow its standard test split with 500 episodes. Each skill in our bank is structured with: a concise title, a principle describing the strategy, and when_to_apply conditions specifying applicability. For ScienceWorld, we follow the official split over task variations. We take the first 10 `train` variations of each task type as $\mathcal{D}_{\text{support}}$, use the `dev` variations as $\mathcal{D}_{\text{query}}$, and report results on 271 unseen `test` variations spanning all 30 task types. The agent worker adopts ReAct-style prompting (Yao et al., 2022b) and retrieves the top-3 skills from the bank via a hybrid scorer that combines BM25 (Robertson and Zaragoza, 2009) with dense cosine similarity over the title, principle, and when_to_apply fields. We run the propose-verify loop for a fixed budget of 10 rounds. At each round, the *Edit Planner* produces K candidate banks, and the Pareto-aware selector uses a utility tolerance $\epsilon = 0.03$ and breaks ties by hypervolume contribution over diversity and coverage. To reduce the cost of repeated counterfactual leave-one-out rollouts required by both *Skill Diagnoser* and the J_{util} computation, we employ a content-addressed counterfactual replay cache that is shared across stages and rounds; see Appendix A.5 for details. All experiments run on a single 8×A800 node.

4.2 Main Results

Comparison with baselines. Table 1 compares SkillBrew against ten training-free baselines on ALFWorld, WebShop, and ScienceWorld, all running on the same frozen Qwen2.5-7B-Instruct worker. SkillBrew achieves the highest overall average on ALFWorld at 59.0%, surpassing the strongest skill-based method Skill-Pro by 9.7% and ReAct by 27.8%. Notably, it also outperforms the append-only Voyager by 12.0%. Relative to ReAct, SkillBrew improves on all six task types, with the largest gains of 47.7% on Heat and 38.9% on Cool. On WebShop, SkillBrew achieves the best results on both metrics, with a Score of 59.3% and a Succ. of 38.4%. On ScienceWorld, SkillBrew again achieves the best results on both metrics, with

Table 1: Main results on ALFWorld (Success Rate, %), WebShop (Score and Success Rate), and ScienceWorld (Score and Success Rate). All methods use Qwen2.5-7B-Instruct as the frozen agent worker. All results are averaged over three random seeds with the standard deviation reported for SkillBrew. **Bold** denotes best results per column; Blue marks the ReAct baseline.

	ALFWorld (Success %)							WebShop		ScienceWorld	
Method	Pick	Look	Clean	Heat	Cool	Pick2	Avg.	Score	Succ.	Score	Succ.
Vanilla Baselines											
Zero-Shot	33.4	21.6	19.3	6.90	2.80	3.20	14.8	26.4	7.80	9.49	5.90
ReAct (Yao et al., 2022b)	48.5	35.4	34.3	13.2	18.2	17.6	31.2	46.2	19.5	11.3	8.13
Memory-Based Methods											
Reflexion (Shinn et al., 2023)	62.0	41.6	44.9	30.9	36.3	23.8	42.7	58.1	28.8	16.2	10.0
Mem0 (Chhikara et al., 2025)	54.0	55.0	26.9	36.4	20.8	7.69	33.6	23.9	2.00	11.8	9.82
MemP (Fang et al., 2026)	54.3	38.5	48.1	56.2	32.0	16.7	41.4	25.3	6.40	18.6	10.8
SimpleMem (Liu et al., 2026a)	64.5	33.3	20.0	12.5	33.3	3.84	29.7	33.2	8.59	11.4	9.94
Skill-Based Methods											
Voyager (Wang et al., 2023)	62.5	61.1	51.6	52.2	28.6	17.6	47.0	50.0	26.4	10.9	9.83
ExpeL (Zhao et al., 2024)	21.0	67.0	55.0	52.0	**71.0**	6.00	46.3	30.9	11.2	13.3	9.76
EvoSkill (Alzubi et al., 2026)	**70.8**	27.8	41.9	39.1	19.0	11.8	37.3	42.2	12.6	19.9	12.9
Skill-Pro (Mi et al., 2026)	66.7	66.7	45.2	43.5	47.6	23.5	49.3	38.7	23.2	17.2	11.4
SkillBrew	**70.8**$\pm$2.4	**72.2**$\pm$3.2	**58.1**$\pm$1.9	**60.9**$\pm$2.5	57.1$\pm$2.7	**29.4**$\pm$3.4	**59.0**$\pm$1.2	**59.3**$\pm$0.8	**38.4**$\pm$0.9	**25.4**$\pm$0.6	**17.2**$\pm$0.7

Table 2: Cross-worker generalization on ALFWorld (Success Rate, %), WebShop (Score and Success Rate), and ScienceWorld (Score and Success Rate). **SkillBrew** is applied on five worker backbones, including open-source and closed-source models. All results are averaged over three random seeds.

		ALFWorld (Success %)							WebShop		ScienceWorld	
Method	**Worker**	Pick	Look	Clean	Heat	Cool	Pick2	Avg.	Score	Succ.	Score	Succ.
ReAct	Qwen2.5-7B-Instruct	48.5	35.4	34.3	13.2	18.2	17.6	31.2	46.2	19.5	11.3	8.13
	Qwen3-4B-Instruct-2507	70.8	55.6	19.4	39.1	23.8	29.4	38.8	49.3	16.8	24.7	18.1
	Qwen3-30B-A3B-Instruct-2507	83.3	61.1	41.9	52.2	33.3	64.7	55.2	43.8	14.0	28.8	18.5
	GPT-4o	81.2	63.1	23.2	56.1	20.2	41.0	46.4	33.3	27.2	57.0	22.5
	Gemini-2.5-Pro	85.2	53.1	67.3	68.2	29.1	59.4	61.8	52.3	38.1	65.2	44.0
SkillBrew	Qwen2.5-7B-Instruct	70.8	72.2	58.1	60.9	57.1	29.4	59.0	59.3	38.4	25.4	17.2
	Qwen3-4B-Instruct-2507	79.2	61.1	54.8	56.5	66.7	41.2	60.4	51.0	36.6	35.9	23.4
	Qwen3-30B-A3B-Instruct-2507	83.3	77.8	80.6	82.6	61.9	58.8	75.4	58.4	56.2	37.2	26.6
	GPT-4o	91.7	88.9	87.1	91.3	81.0	88.2	88.1	75.0	65.0	60.3	30.1
	Gemini-2.5-Pro	95.8	94.4	90.3	87.0	76.2	94.1	89.6	82.4	72.0	68.6	50.3

a Score of 25.4% and a Succ. of 17.2%, surpassing the strongest baseline EvoSkill by 5.5% and 4.3% and ReAct by 14.1% and 9.1%. In WebShop and ScienceWorld, the Score credits partial progress while Succ. requires satisfying every requirement, so the substantial gains on Succ. indicate that the curated skills carry episodes through to full completion, matching every required attribute of a purchase or completing an entire experimental procedure, rather than stopping at near misses that earn only partial reward.

Cross-worker generalization. Table 2 extends the evaluation to different worker backbones, including three Qwen open-source models (Yang et al., 2024, 2025) and two closed-source models, GPT-4o (Hurst et al., 2024) and Gemini-2.5-Pro (Comanici et al., 2025). SkillBrew improves over the ReAct baseline on every backbone and on all three benchmarks, with ALFWorld Avg. gains ranging from 20.2% on Qwen3-30B-A3B-Instruct-2507 to 41.7% on GPT-4o. On ScienceWorld, where GPT-4o and Gemini-2.5-Pro already reach Scores of 57.0 and 65.2 under ReAct, SkillBrew still improves Succ. on every backbone, with gains ranging from 5.3% to 9.1%. The improvements are particularly pronounced on closed-source workers. On GPT-4o, ALFWorld Avg. rises from 46.4 to 88.1 and WebShop Succ. from 27.2 to 65.0, a gain of 37.8%. On ALFWorld and WebShop, the much smaller Qwen3-4B-Instruct-2507 worker after curation reaches an ALFWorld Avg. of 60.4% and a WebShop Succ. of 36.6%, approaching the 61.8% and 38.1% of Gemini-2.5-Pro without cura-

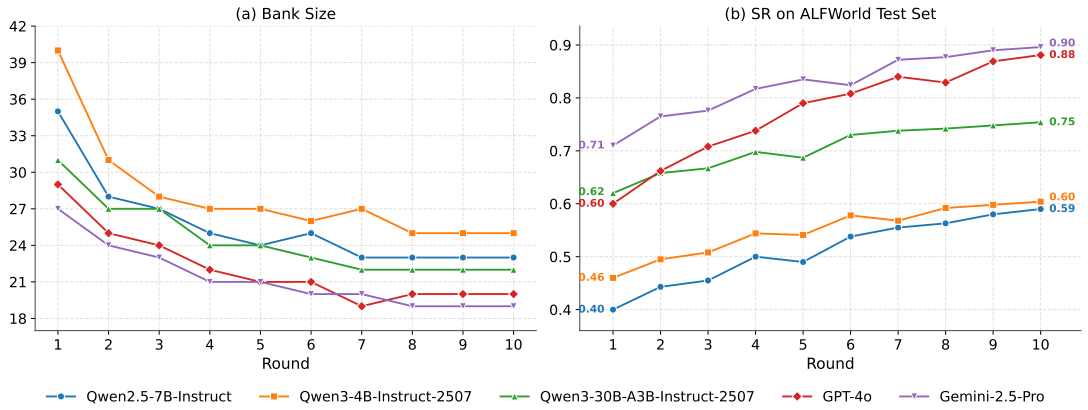

Figure 3: Curation loop dynamics of SkillBrew across five worker backbones on ALFWorld. (a) Bank size across rounds. (b) Success rate on ALFWorld test set across rounds.

Table 3: Bank transfer across workers on ALFWorld (Avg. SR, %). **Bold** denotes best results.

Bank Source	Agent Worker		
	Qwen3-4B Instruct-2507	Qwen3-30B-A3B Instruct-2507	GPT-4o
no bank (ReAct)	38.8	55.2	46.4
Qwen3-4B Instruct-2507	**60.4**	71.6	78.4
Qwen3-30B-A3B Instruct-2507	52.2	**75.4**	82.8
GPT-4o	53.7	73.9	**88.1**

Table 4: Ablation on the objective formulation. Each row uses a different subset of three objectives, while keeping the rest of the curation pipeline fixed. **Bold** denotes best results.

Objectives	ALFWorld	WebShop	
	Avg.	Score	Succ.
J_{util}	45.8	48.2	28.4
$J_{\text{util}} + J_{\text{div}}$	51.4	47.6	36.5
$J_{\text{util}} + J_{\text{cov}}$	52.6	51.0	37.6
$J_{\text{util}} + J_{\text{div}} + J_{\text{cov}}$	**59.0**	**59.3**	**38.4**

tion, suggesting that a well-curated skill bank can partially compensate for model size.

4.3 Analysis

4.3.1 Curation Loop Dynamics

Figure 3 traces the dynamics of skill bank over the curation loop across five worker backbones on ALFWorld. The bank size contracts from larger cold-start values to compact banks within 9–10 rounds and stabilizes thereafter, indicating that the curation process converges across all workers rather than growing unboundedly. The test success rate rises throughout the loop on every worker, with small round-to-round fluctuations but clear overall gains. Even at round 1, where the bank only uses the cold-start output of the *Skill Distiller*, the test success rate is already substantially above the ReAct baseline of each worker, confirming that the cold-start design provides a strong starting point that the subsequent curation loop further improves upon. Representative skills from the final curated bank for each benchmark are shown in Appendix B.

4.3.2 Ablation Study

Bank transferability. Table 3 examines whether a curated bank for one worker remains effective when applied to a different worker at inference time. We curate three skill banks with SkillBrew, each under a different source worker, and apply each bank to all three inference workers. Every cell substantially exceeds the ReAct baseline without retrieval. Diagonal cells, where each worker retrieves from its own bank, yield the best result for that worker at 60.4%, 75.4%, and 88.1%, indicating that curation specializes the bank to its source worker. Cells away from the diagonal remain close behind in both directions: a bank curated by the smallest Qwen3-4B-Instruct-2507 worker lifts GPT-4o from 46.4% to 78.4%, only 9.7% below GPT-4o's own result; conversely, a bank curated by GPT-4o lifts Qwen3-4B-Instruct-2507 worker from 38.8% to 53.7%, only 6.7% below its own result. This suggests that the curated skills encode transferable procedural knowledge rather than artifacts of a particular worker, so a skill bank can be reused across workers in either direction with little loss.

Table 5: Ablation on the edit operations. Each row uses a different subset of the three edit operations. **Bold** denotes best results.

Edit Operation			ALFWorld	WebShop	
ADD	REWRITE	REMOVE	Avg.	Score	Succ.
✓			47.0	50.0	26.4
✓	✓		53.5	51.0	37.6
✓		✓	48.3	49.5	34.5
✓	✓	✓	**59.0**	**59.3**	**38.4**

Effect of objective. Table 4 ablates the contribution of each objective by varying which terms enter the Pareto-aware selector while keeping curation pipeline fixed. Using J_{util} alone yields a success rate of 45.8% on ALFWorld and 28.4% on WebShop, trailing the full objective by 13.2% and 10.0% respectively. Compared to using J_{util} alone, adding J_{div} or adding J_{cov} improves the success rate. The two objectives are complementary: J_{div} promotes diversity and prevents redundancy in skill bank, while J_{cov} penalizes banks that bloat with dormant or rarely retrieved skills. Combining both reaches the best results on both benchmarks.

Effect of edit operations. Table 5 ablates the three edit operations available to the *Edit Planner*. With ADD alone, the bank grows monotonically without ever pruning or refining, leaving dormant or harmful skills to crowd out retrieval slots; the success rate reaches only 47.0% on ALFWorld and 26.4% on WebShop. The ADD+REWRITE setting lifts ALFWorld to 53.5% because the majority of skills the Diagnoser flags are salvageable, and refinement recovers their value rather than discarding it. ADD+REMOVE setting lifts ALFWorld only to 48.3%, lower than ADD+REWRITE because the operation is too coarse: every flagged skill must be discarded directly, including partially correct ones whose strategy could have been salvaged and refined by a small edit. Combining all three reaches 59.0% on ALFWorld and 38.4% on WebShop, indicating that the two operations play complementary roles: REWRITE is the default rescue path that handles the bulk of flagged skills, while REMOVE is reserved for the minority whose underlying strategy cannot be repaired by any edit.

5 Conclusion

We present SkillBrew, a training-free framework that curates skill banks for LLM agents through bank-level optimization. Instead of judging each skill independently or continuously adding new skills, SkillBrew treats the entire skill bank as the optimization object and balances utility, diversity, and coverage under a Pareto-aware selection process. Experiments on ALFWorld, WebShop, and ScienceWorld show that this bank-level curation leads to more diverse, transferable, and effective skill repositories, improving agent performance across both open-source and closed-source workers. These results suggest that future self-improving agents should optimize skill banks as global knowledge systems, rather than manage them as isolated skills.

Limitations

One limitation of our framework is that the agent remains frozen during skill bank optimization, leaving joint co-adaptation between the agent and the evolving bank unexplored. Future work could investigate recursive co-training or reinforcement learning frameworks that jointly optimize both components. In addition, although our content-addressed cache substantially reduces the cost of counterfactual rollouts, the method still relies on repeated rollouts and leave-one-out replay, which may become expensive at larger scales; a more efficient method could further improve scalability. Finally, the current formulation assumes relatively static task distributions, and extending bank curation to open-world and continually shifting environments remains an important direction. Nevertheless, we believe this work introduces a new perspective: skill banks should be treated not as append-only memory repositories, but as globally optimized and continually refined objects, opening new directions for scalable and lifelong agent systems.

Acknowledgments

This research was partially supported by National Natural Science Foundation of China (No.62502404), Hong Kong Research Grants Council (Research Impact Fund No.R1015-23, Collaborative Research Fund No.C1043-24GF, RGC Research Fellow Scheme No.RFS2627-1S03, General Research Fund No.11218325, No.11212926), Institute of Digital Medicine of City University of Hong Kong (No.9229503).

References

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. 2026. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*.

Anthropic. 2025. Agent skills. https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview.

Yair Censor. 1977. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, 4(1):41–59.

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

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2026. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 17490–17502.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. 2026a. Simplemem: Efficient lifelong memory for llm agents. *arXiv preprint arXiv:2601.02553*.

Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. 2026b. Skillforge: Forging domain-specific, self-evolving agent skills in cloud technical support. *arXiv preprint arXiv:2604.08618*.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. 2026. Skill-pro: Learning reusable skills from experience via non-parametric ppo for llm agents. *arXiv preprint arXiv:2602.01869*.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. 2026. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*.

OpenAI. 2026. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4/.

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

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11298.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, and 1 others. 2026. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022a. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022b. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, and 1 others. 2026. Coevoskills: Self-evolving agent skills via co-evolutionary verification. *arXiv preprint arXiv:2604.01687*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, and 1 others. 2026. Memento-skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*.

A Implementation Details

A.1 Datasets

ALFWorld. ALFWorld (Shridhar et al., 2020) is a text-based embodied benchmark that spans six household task families. We randomly sample 200 episodes from the official `train` split as the support set $\mathcal{D}_{\text{support}}$, and use the full `valid_seen` split as the query set $\mathcal{D}_{\text{query}}$ and the full `valid_unseen` split as the test set. Sampling uses random seed 42.

WebShop. WebShop (Yao et al., 2022a) is a simulated online shopping environment with a fixed product catalog and one-million-scale natural language goals; the standard test split contains 500 goals, which we use as the test set. We additionally sample 500 and 250 goals from WebShop training and evaluation pool (goal ids ≥ 500) as $\mathcal{D}_{\text{support}}$ and $\mathcal{D}_{\text{query}}$ respectively, with seed 42. The support, query, and test sets are pairwise disjoint.

ScienceWorld. ScienceWorld (Wang et al., 2022) is an interactive text environment for elementary science experiments that are officially split into `train`, `dev`, and `test`. The `dev` and `test` variations instantiate materials and organisms unseen during `train`. We use the first 10 `train` variations of each task type to form $\mathcal{D}_{\text{support}}$, the `dev` variations form $\mathcal{D}_{\text{query}}$, and the test set consists of the first 10 `test` variations of each task type with 271 instances in total, since several task types have fewer than 10 test variations.

A.2 LLM Settings

The agent worker is prompted in ReAct style (Yao et al., 2022b). On ALFWorld, the worker is given a budget of 50 steps per episode and conditions on the full history at every generation step. On WebShop, the worker is given 15 steps per episode and conditions on the most recent one step as history. The maximum generation length per step is 2,048 tokens for all agent workers. For *Skill Distiller*, *Skill Diagnoser*, and *Edit Planner*, we all use GPT-5.4 (OpenAI, 2026). Their prompts are shown in Appendix C.

A.3 Skill Retrieval

The agent retrieves the top-3 most relevant skills from skill bank using a hybrid scorer that combines BM25 (Robertson and Zaragoza, 2009) with dense cosine similarity. Concretely, the score of a skill s relative to a query q is

$$\text{score}(q, s) = 0.30 \cdot \overline{\text{BM25}}(q, s) + 0.70 \cdot \cos(\mathbf{e}_q, \mathbf{e}_s),$$

where $\overline{\text{BM25}}$ denotes min-max normalized BM25 scores. The skill text used for retrieval is the combination of three fields: `title`, `principle`, and `when_to_apply`. Skills with a combined score below 0.30 are dropped from the retrieval result. We use text-embedding-3-small for generating skill embeddings.

A.4 Curation Loop and Pareto-Aware Selector

We run the propose-then-verify loop for a fixed budget of 10 rounds. Within each round, the *Skill Distiller*, *Skill Diagnoser*, and *Edit Planner* are invoked in this order, and each candidate bank proposed by the Edit Planner is evaluated by rolling out the worker on $\mathcal{D}_{\text{query}}$ to estimate J_{util}, J_{div}, and J_{cov}. The Pareto-aware selector uses a hypervolume contribution criterion with lex priority on J_{util}: candidates whose J_{util} falls within a tolerance of $\epsilon = 0.03$ of the highest feasible J_{util} are admitted to the tie pool, and ties are broken by hypervolume contribution over J_{div} and J_{cov}. The previous bank is augmented to the candidate set as a null candidate. If it wins the lex-Pareto tie-break by hypervolume contribution, the bank carries forward unchanged to the next round.

Per-round utility gap. The per-round ϵ-utility safeguard that follows from Eq. 14 bounds the utility drop of a single round by ϵ, so the formal worst case after T rounds is $T\epsilon$. To quantify how much of this slack is actually consumed, we measure the per-round utility gap

$$\delta_i = u_{\max}^{(i)} - J_{\text{util}}\big(\mathcal{B}^{(i+1)}\big), \tag{15}$$

where $u_{\max}^{(i)}$ is the largest J_{util} on the Pareto front of round i, so that δ_i is the utility given up in that round relative to the best candidate available. By construction $0 \le \delta_i \le \epsilon$, since the selected bank is drawn from Tied. Moreover, since the null candidate is always available, $u_{\max}^{(i)} \ge J_{\text{util}}(\mathcal{B}^{(i)})$, so δ_i upper-bounds the actual utility change between consecutive rounds. Across the curation rounds on ALFWorld with the Qwen2.5-7B-Instruct worker, the mean gap is 0.004, the maximum gap is 0.015, and 40% of the rounds have $\delta_i = 0$ because either the highest-utility candidate or the null candidate is selected. The realized cumulative drop is therefore bounded by $\sum_i \delta_i \approx 0.04$, far below the worst case $T\epsilon = 0.3$. This indicates that $\epsilon = 0.03$ mainly delimits an admissible trade-off region within which diversity and coverage can be improved, rather than inducing a fixed utility loss at every iteration.

Per-round objective trajectories. Figure 4 traces the three objectives across the ten curation rounds. The trajectories exhibit the trade-off predicted by the principles of §3.2. On ALFWorld, J_{util} rises from 0.052 at cold start to 0.423 and J_{div} from 0.153 to 0.343, while J_{cov} decreases from 0.716 to 0.583 and then flattens. The selector prunes redundant skills, which concentrates utility and raises structural diversity at the price of coverage on the long tail, consistent with the bank contracting from 35 to 23 skills in Table 8. WebShop exhibits the same pattern more sharply, where J_{util} and J_{div} reach 0.623 and 0.548 while J_{cov} falls to 0.507, so the ordering at cold start is reversed and coverage ends below both other objectives. In both runs J_{util} is not strictly monotone, and the largest single round decrease is 0.007, well inside the tolerance $\epsilon = 0.03$, which is precisely the behavior that the per-round ϵ-utility safeguard admits. The round-10 values on ALFWorld coincide with the default $\epsilon = 0.03$ row of Table 6, since that setting is the one used in all other experiments.

Effect of the utility tolerance ϵ. The tolerance ϵ in Eq. 14 is the single knob that controls how much utility slack the selector may spend on diversity and coverage. As $\epsilon \to 0$ the selection collapses into strict utility maximization and the two regularizers lose influence, whereas a larger ϵ admits a wider tie pool and lets them shape the bank more strongly. Table 6 sweeps ϵ on ALFWorld. The two objectives move in opposite directions exactly as the formulation predicts. J_{util} decreases monotonically from 0.452 to 0.396 while J_{div} increases monotonically from 0.324 to 0.362, confirming that ϵ trades query set utility for bank diversity. Downstream success rate, however, is not monotone in ϵ and peaks at the default $\epsilon = 0.03$. Notably, $\epsilon = 0.01$ attains the highest J_{util} on the query set yet a lower test success rate than $\epsilon = 0.03$, which indicates overfitting, as a utility-greedy selector fits the query trajectories more tightly but transfers less well to unseen environments. At the other extreme, $\epsilon = 0.10$ sacrifices too much utility for structure and degrades to 56.7. J_{cov} stays comparatively flat and attains its minimum at the default setting, consistent with the bank contraction shown in Figure 3, where the curated bank trades away only redundant coverage. Diversity and coverage therefore convert a small amount of query-set utility into better generalization rather than acting as proxies for it, and this benefit is realized over a moderate range of ϵ rather than at either extreme.

Table 6: Effect of the utility tolerance ϵ on ALFWorld with the Qwen2.5-7B-Instruct worker. † marks the default setting used in all other experiments.

ϵ	J_{util}	J_{div}	J_{cov}	Test SR (%)
0.01	0.452	0.324	0.586	56.7
0.03†	0.423	0.343	0.583	**59.0**
0.05	0.412	0.358	0.594	57.5
0.10	0.396	0.362	0.608	56.7

A.5 Counterfactual Replay Cache

The propose-then-verify loop requires counterfactual leave-one-out rollouts in two places: on support set $\mathcal{D}_{\text{support}}$, where the Skill Diagnoser compares with and without each retrieved skill to issue KEEP / REWRITE / REMOVE verdicts, and on query set $\mathcal{D}_{\text{query}}$, where each candidate bank's J_{util} is computed by the same leave-one-out procedure.

To improve efficiency across both stages, we maintain a single content-addressed cache shared by all rollouts. Each entry is keyed by a SHA-256 hash of the task id and the canonical content (`title`, `principle`, `when_to_apply`) of ev-

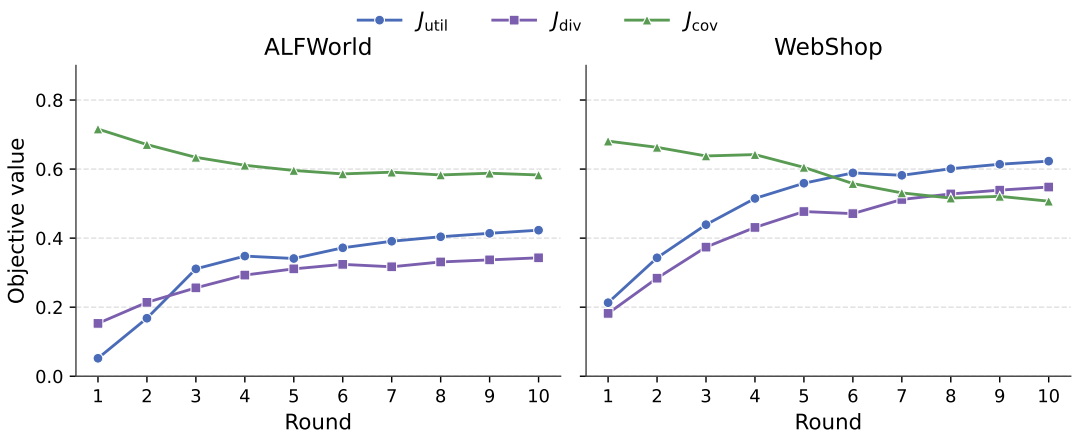

Figure 4: Per-round trajectories of the three objectives on ALFWorld (left) and WebShop (right) with the Qwen2.5-7B-Instruct worker.

ery retrieved skill, ordered by retrieval rank, so any change visible to the worker is also visible to the key. Entries also carry a version tag derived from the worker configuration; whenever the worker model or its prompt template changes, all cached entries are silently invalidated, preventing stale rollouts from polluting either the Diagnoser verdicts or the J_{util} signal. The effect of this cache on the overall cost is reported in §A.6.

A.6 Cost Analysis

We report the computational cost of a full curation run to characterize the practical overhead of SkillBrew. All measurements are taken on a single 8×A800 node with the Qwen2.5-7B-Instruct worker over the full budget of $T = 10$ rounds on ALFWorld. The agent worker is served locally, so there is no API cost, while the *Skill Distiller*, *Skill Diagnoser*, and *Edit Planner* are the only paid components. Across our three open-source workers, a complete run costs between 8.8 and 11.2 USD and finishes in under eight hours.

Component breakdown. Table 7 decomposes a full run by pipeline stage. Agent rollouts dominate the wall-clock. The skill rollouts, in which the worker solves tasks with the skills retrieved from the bank, together with the leave-one-out replay used for counterfactual credit assignment, account for roughly 81% of the total time. Because both run locally on the frozen worker, they contribute nothing to the monetary cost. The three LLM-driven curation stages consume only 19% of the wall-clock yet account for the entire 11.2 USD API cost, which is itself modest relative to any training-based methods.

Table 7: Cost breakdown of a full 10-round curation on ALFWorld with the Qwen2.5-7B-Instruct worker.

Stage	Executor	Wall-clock		API cost	
		min	%	USD	%
Skill rollouts	Worker	287.4	61	0.0	0
Leave-one-out replay	Worker	96.8	20	0.0	0
Skill Distiller	GPT-5.4	44.6	9	2.3	21
Skill Diagnoser	GPT-5.4	31.7	7	4.5	40
Edit Planner	GPT-5.4	13.3	3	4.4	39
Total	–	**473.8**	**100**	**11.2**	**100**

Per-round cost. Table 8 reports how the cost evolves across rounds. Two trends are worth noting. First, after the early rounds the per-round cost decreases monotonically, from 87.4 minutes at round 2 to 33.4 minutes at round 10. Because the selector prunes as well as adds, the bank contracts from 35 skills at cold start to a stable size of 23, so the set of retrieved skills that must be replayed shrinks rather than grows. Append-only methods have the opposite profile, since every round enlarges the set of skills whose contribution must be assessed. Second, the number of candidate banks K proposed per round stays small, between 7 and 11 from round 2 onward, and candidates are evaluated in parallel on the local worker, so a larger K adds amortizable wall-clock but no additional cost.

Effect of the replay cache. Because the K candidate banks proposed by the Edit Planner within a round differ by only a few edits, and the winning bank further carries forward to the next round, most leave-one-out retrievals coincide across candidates and across rounds, so the content-addressed cache of §A.5 is hit often. We observe a cache hit rate

Table 8: Per-round curation cost on ALFWorld with the Qwen2.5-7B-Instruct worker. K denotes the number of candidate banks proposed in that round.

Round	Bank size	K	Wall-clock (min)		
			w/ cache	w/o cache	Δ
1	35	–	19.4	19.4	0.0
2	28	11	87.4	229.7	142.3
3	27	9	70.3	197.6	127.3
4	25	11	58.1	231.4	173.3
5	24	9	49.6	199.2	149.6
6	25	7	45.2	165.8	120.6
7	23	9	40.3	196.8	156.5
8	23	8	36.4	181.7	145.3
9	23	10	33.7	214.3	180.6
10	23	8	33.4	182.4	149.0
Total	–	–	**473.8**	**1818.3**	**1344.5**

above 60% from the second round onwards, which translates into a substantial saving in practice: as the last two columns of Table 8 show, disabling the cache raises the total cost of a full run from 473.8 to 1818.3 minutes, an overall speedup of roughly $3.8\times$ and 1344.5 minutes saved over a single run.

B Skill Examples

Figure 5 and Figure 6 present representative skills from the final curated bank on ALFWorld and WebShop, respectively; Figure 7 additionally traces a single skill before and after a REWRITE verdict from the *Skill Diagnoser*.

C Prompts

Figures 8 and 9 show the two-stage prompt used by the *Skill Distiller*: the first analyzes failure trajectories to identify missing capabilities, while the second synthesizes candidate ADD skills targeting those gaps. Figure 10 shows the prompt of the *Skill Diagnoser*, which assigns KEEP, REWRITE, or REMOVE verdicts to each retrieved skill from its factual and counterfactual outcomes. Figure 11 shows the prompt of the *Edit Planner*. Rather than mechanically applying each edit verdict, the planner conditions on the full current bank together with all candidate edit pools, reasoning at the bank level about which combinations of edits compose into K candidate banks.

Algorithm 1: Inner Loop: Skill Edit Proposal from Support Trajectories

Input: Current skill bank $\mathcal{B}^{(i)}$, support set $\mathcal{D}_{\text{support}}$, frozen agent π_{agent}, retriever RET, number of candidates K
Output: Candidate bank set $\mathcal{K}^{(i)} = \{\mathcal{B}_k^{(i+1)}\}_{k=1}^{K}$

// Roll out and partition trajectories by outcome and retrieval status
$\mathcal{T}_{\bar{R}}^{+} \leftarrow \emptyset$; $\mathcal{T}_{\bar{R}}^{-} \leftarrow \emptyset$; $\mathcal{T}_{R}^{+} \leftarrow \emptyset$; $\mathcal{T}_{R}^{-} \leftarrow \emptyset$
foreach $t \in \mathcal{D}_{support}$ **do**
  $R_t \leftarrow \text{RET}(\mathcal{B}^{(i)}, t)$;
  $\tau_t \leftarrow \texttt{Rollout}(\pi_{\text{agent}}, t, R_t)$
  if $\text{SUCC}(\tau_t)$ ***and*** $R_t = \emptyset$ **then**
    $\mathcal{T}_{\bar{R}}^{+} \leftarrow \mathcal{T}_{\bar{R}}^{+} \cup \{\tau_t\}$
  else if $\text{FAIL}(\tau_t)$ ***and*** $R_t = \emptyset$ **then**
    $\mathcal{T}_{\bar{R}}^{-} \leftarrow \mathcal{T}_{\bar{R}}^{-} \cup \{\tau_t\}$
  else if $\text{SUCC}(\tau_t)$ ***and*** $R_t \neq \emptyset$ **then**
    $\mathcal{T}_{R}^{+} \leftarrow \mathcal{T}_{R}^{+} \cup \{(t, R_t, \tau_t)\}$
  else
    $\mathcal{T}_{R}^{-} \leftarrow \mathcal{T}_{R}^{-} \cup \{(t, R_t, \tau_t)\}$
// Distill Add candidates from no-retrieval failures, with no-retrieval successes as positive references
$\mathcal{E}_{\text{add}} \leftarrow \texttt{Distill}(\mathcal{T}_{\bar{R}}^{-}, \mathcal{T}_{\bar{R}}^{+})$
// Collect leave-one-out evidence pairs per retrieved skill
$\mathcal{P}_s \leftarrow \emptyset$ **for each** s
foreach $(t, R_t, \tau_t) \in \mathcal{T}_{R}^{+} \cup \mathcal{T}_{R}^{-}$ **do**
  foreach $s \in R_t$ **do**
    $\tilde{\tau}_{t \setminus s} \leftarrow \texttt{Replay}(\pi_{\text{agent}}, t, R_t \setminus \{s\})$
    $\mathcal{P}_s \leftarrow \mathcal{P}_s \cup \{(\tau_t, \tilde{\tau}_{t \setminus s})\}$
// Diagnose each skill once over its full pool of counterfactual pairs
$\mathcal{E}_{\text{keep}} \leftarrow \emptyset$; $\mathcal{E}_{\text{rewrite}} \leftarrow \emptyset$; $\mathcal{E}_{\text{remove}} \leftarrow \emptyset$
foreach s ***with*** $\mathcal{P}_s \neq \emptyset$ **do**
  $v_s \leftarrow \texttt{Diagnose}(s, \mathcal{P}_s)$
  if $v_s = \text{KEEP}$ **then**
    $\mathcal{E}_{\text{keep}} \leftarrow \mathcal{E}_{\text{keep}} \cup \{s\}$
  else if $v_s = \text{REWRITE}$ **then**
    $\mathcal{E}_{\text{rewrite}} \leftarrow \mathcal{E}_{\text{rewrite}} \cup \{(s, \mathcal{P}_s)\}$
  else
    $\mathcal{E}_{\text{remove}} \leftarrow \mathcal{E}_{\text{remove}} \cup \{s\}$
// Compose edits into candidate banks; Keep set is passed as protected slots
$\mathcal{K}^{(i)} \leftarrow \texttt{PlanEdit}\left(\mathcal{B}^{(i)}, \mathcal{E}_{\text{add}}, \mathcal{E}_{\text{rewrite}}, \mathcal{E}_{\text{remove}}, \mathcal{E}_{\text{keep}}, K\right)$
return $\mathcal{K}^{(i)}$

Algorithm 2: Outer Loop: Pareto-Aware Skill Bank Curation

Input: Support set $\mathcal{D}_{\text{support}}$, query set $\mathcal{D}_{\text{query}}$, frozen agent π_{agent}, retriever RET, max rounds T, number of candidates K, utility tolerance ϵ
Output: Curated skill bank $\mathcal{B}^{\star}$

// Initialize skill bank from no-retrieval trajectories via the Skill Distiller
$\mathcal{B}^{(0)} \leftarrow \texttt{ColdStart}(\pi_{\text{agent}}, \mathcal{D}_{\text{support}})$
for $i = 0$ **to** $T - 1$ **do**
  // Propose candidate banks from support-set evidence
  $\mathcal{K}^{(i)} \leftarrow \texttt{InnerLoop}\left(\mathcal{B}^{(i)}, \mathcal{D}_{\text{support}}, \pi_{\text{agent}}, \text{RET}, K\right)$
  // Include the current bank as a null candidate to enable a no-edit decision
  $\mathcal{C}^{(i)} \leftarrow \mathcal{K}^{(i)} \cup \{\mathcal{B}^{(i)}\}$
  // Evaluate utility, diversity, and coverage on query set
  foreach $\mathcal{B} \in \mathcal{C}^{(i)}$ **do**
    $J_{\text{util}}(\mathcal{B}) \leftarrow \texttt{Eval}_{\texttt{util}}(\mathcal{B}, \mathcal{D}_{\text{query}}, \pi_{\text{agent}}, \text{RET})$ // requires LOO rollouts
    $J_{\text{div}}(\mathcal{B}) \leftarrow \texttt{Eval}_{\texttt{div}}(\mathcal{B})$ // structural, no rollouts
    $J_{\text{cov}}(\mathcal{B}) \leftarrow \texttt{Eval}_{\texttt{cov}}(\mathcal{B}, \mathcal{D}_{\text{query}}, \text{RET})$ // retrieval only
    $\Phi(\mathcal{B}) \leftarrow \left(J_{\text{util}}(\mathcal{B}), J_{\text{div}}(\mathcal{B}), J_{\text{cov}}(\mathcal{B})\right)$
  // Construct the non-dominated Pareto front
  $\mathcal{F}^{(i)} \leftarrow \texttt{ParetoFront}\left(\{(\mathcal{B}, \Phi(\mathcal{B})) : \mathcal{B} \in \mathcal{C}^{(i)}\}\right)$
  // Prioritize utility, then break ties by diversity-coverage hypervolume
  $u_{\max} \leftarrow \max_{\mathcal{B} \in \mathcal{F}^{(i)}} J_{\text{util}}(\mathcal{B})$
  $\mathcal{M}^{(i)} \leftarrow \left\{\mathcal{B} \in \mathcal{F}^{(i)} : J_{\text{util}}(\mathcal{B}) \geq u_{\max} - \epsilon\right\}$
  $\mathcal{B}^{(i+1)} \leftarrow \arg\max_{\mathcal{B} \in \mathcal{M}^{(i)}} \texttt{Hypervolume}\left(J_{\text{div}}(\mathcal{B}), J_{\text{cov}}(\mathcal{B})\right)$
  // if the null candidate wins, $\mathcal{B}^{(i+1)} = \mathcal{B}^{(i)}$

$\mathcal{B}^{\star} \leftarrow \mathcal{B}^{(T)}$
return $\mathcal{B}^{\star}$

Title: Systematic heat-task search

Principle:
1) If the object is not immediately visible, first check likely containers such as fridge, cabinet, and microwave; open them before examining when needed.

2) If the object is still not found, check countertop, diningtable, shelf, and coffeemachine before revisiting earlier places.

3) Once the object is found, take it, go to microwave, open microwave if needed, and heat it.

4) After heating, go directly to the destination receptacle and put it there; avoid looping between fridge, cabinet, and microwave or repeating look/inventory before checking new locations.

When to apply:
When the task requires heating the object before placing it in a destination receptacle, and the object is not found in the first one or two inspected locations.

Title: Use put at destination

Principle:
1) Apply this skill only when the target object is already in your inventory; otherwise return to object search rather than troubleshooting placement.

2) Go to the task-specified destination; if it is a closable receptacle such as a drawer, cabinet, fridge, or safe, open it before placing.

3) Use the environment action that places held objects into the destination, and do not switch verbs based on natural language preference.

4) If a placement attempt returns Nothing happens, verify that you are at the destination and that it is open if closable, then retry once; if it still fails, reassess whether you have the correct object and location rather than looping on the same action.

When to apply:
When a pick_and_place_simple task reaches the stage where the object is already in hand, or when a placement attempt returns Nothing happens.

Title: Use lamp before inspect

Principle:
1) For tasks that say to look at or examine an object with or under a desklamp, first find a surface that has a desklamp and turn the lamp on.

2) If the target object is already on that same surface, take or otherwise interact with the target there after the lamp is on.

3) If the target object is elsewhere, pick it up and bring it to the lit desklamp location, then stop by returning to that lit surface while holding the object.

When to apply:
When the task says to look at or examine the object with or under the desklamp, especially if you have already found the object but not the desklamp.

Figure 5: Example skills from the curated bank on ALFWorld.

Title: Core noun first, modifiers second

Principle:
1) Construct the initial search query by placing the core product noun first, such as “running shoes”, “wireless mouse”, or “hand cream”, followed by one or two of the most restrictive attributes named in the goal, such as color, size, gender, or material, and end with any hard numeric constraints such as price.

2) Omit aesthetic descriptors, brand qualifiers, and packaging adjectives unless they are mentioned in the goal text, since these tend to narrow the result page to a small or empty set.

3) If the goal mentions multiple modifiers of the same kind, such as two colors, include only the first one in the initial query and reserve the second for the option-selection step.

When to apply:
When a new shopping goal has just been issued and no search has been performed yet, especially when the goal contains a noun phrase followed by several attribute modifiers.

Title: Commit to Buy after all required attributes are set

Principle:
1) Issue the Buy action only after every option dimension named in the goal has been selected with a value that matches verbatim.

2) Before clicking Buy, re-read the goal once and cross-check each named attribute against the current option-selection state on the page; do not rely on memory of earlier action results.

3) If any goal-named attribute is still unset or mismatched, do not click Buy; instead, select that attribute on the current page if a matching option exists, otherwise return to search.

4) Do not click Buy as a default fallback when stuck; an incorrect or incomplete purchase ends the episode with a partial match instead of full success.

When to apply:
When all goal-named option dimensions appear to be selected on the current product page and the next action is between Buy and further inspection.

Title: Verify hard constraints before any option click

Principle:
1) Before clicking any option on a product page, scan the page text and confirm two hard constraints from the goal in this order: (a) the product's category matches the requested noun, and (b) the price satisfies any numeric upper bound in the goal.

2) If either hard constraint is violated, return to the search results page or issue a new search; do not select options or click Buy on a product that fails a hard constraint.

3) If both constraints are satisfied, proceed to the option-selection step; do not re-verify constraints on every subsequent action within the same product page.

When to apply:
When the agent has just opened a product detail page from search results, before selecting any size, color, scent, or quantity option.

Figure 6: Example skills from the curated bank on WebShop.

Round 1

title: Systematic room search

principle:
1) First go to the destination receptacle and examine it so you know where to return.

2) If the object is not there, search new locations in a fixed sweep: go to shelf, sidetable, coffeetable, desk, dresser, sofa, armchair, cabinet, drawer, and tvstand; at each place examine it, and if it is a cabinet or drawer then open it before examining inside.

3) As soon as you see the object, take it, go to the destination receptacle, and put it there; do not keep using look or bounce back to already checked places until all unvisited locations are checked.

when_to_apply:
When a pick_and_place_simple task starts and the object is not visible at the first checked location, or when you notice repeated visits to the same receptacle without having taken the object.

Round 3

title: Systematic room search

principle:
1) First identify and visit the destination receptacle named in the task so you know where to return; if it is openable, open it before using it.

2) If the target object is not already visible there, search systematically through unvisited receptacles in the current room, prioritizing plausible surfaces and containers you can actually access from your current view; when a receptacle is openable, such as a drawer, cabinet, safe, fridge, or microwave, open it before examining inside.

3) Keep track of what you have already checked and prefer new locations over revisiting old ones; only revisit a place if new information suggests it is relevant.

4) As soon as you see the target object, take it, go to the destination receptacle, and place it there; if placement fails, first move to the destination receptacle and then place it again.

5) Do not keep using look or examine repeatedly on the same place without gaining new information.

when_to_apply:
When a pick_and_place_simple task starts and the object is not visible at the first checked location, or when you notice repeated visits to the same receptacle without having taken the object.

Round 7

title: Systematic room search

principle:
1) At the start, identify both the target object and the destination receptacle named in the task, but do not assume you must visit the destination first; if the target is already visible nearby, take it first.

2) Otherwise, search the current room systematically through unvisited receptacles, preferring nearby visible locations before remote or already-checked ones; open any receptacle that could hold the target or the destination before judging that location.

3) Keep track of where you have already searched; do not bounce between the same receptacles or repeat look or examine without new information.

4) Once you find the target, take it and go directly to the destination to place it; if placement fails, re-approach the destination and retry once.

when_to_apply:
When a pick_and_place_simple task starts and the object is not visible at the first checked location, or when you notice repeated visits to the same receptacle without having taken the object.

Figure 7: An example of REWRITE.

```
You are analyzing a FAILED episode in {benchmark_name}, {env_description}. No skill was retrieved — the agent had no guidance.

Task: "{task_description}"
Full trajectory ({num_steps} steps): {trajectory_text}

Identify the primary failure pattern of this trajectory: locate the step where things first went wrong, summarize the failure, and
describe what the agent should have done instead.


Output a JSON object:
{
"error_step": <step index where things first went wrong>,
"failure_reason": "<one concise sentence describing the primary failure>",
"what_should_have_done": "<one sentence: what the agent should have done instead>"
}
Return ONLY the JSON object.
```

Figure 8: Prompt of the *Skill Distiller* for failure analysis.

```
You are a skill library curator for an LLM agent. Your task is to distill actionable skills from the failed no-retrieval trajectories below,
using the successful no-retrieval trajectories as positive reference material.


=== Failed Trajectories ({n_failed}) ===
Each entry below is one failed no-retrieval trajectory, paired with a short analysis from the failure-analysis pass (the step where
things first went wrong, a one-sentence failure reason, and what the agent should have done instead). Cluster these failures into
coherent sub-patterns yourself.
{failed_episodes_with_trajectories_and_analyses}


=== Successful Reference Trajectories ({n_success}) ===
These agents succeeded WITHOUT skills. Their action sequences are concrete positive templates — pay attention to step ordering,
entity visits, and recovery patterns. Not every reference matches every failure; extract the generalizable pattern per cluster.
{successful_episodes_with_trajectories}


=== Your Task ===
For each genuinely distinct sub-failure mechanism you identify in the failed trajectories, produce one skill that targets that sub-
pattern. Each skill must include ordered sub-steps grounded in the successful reference trajectories and explicitly state what to
avoid. Principle format: short sentences, numbered steps preferred.


=== Output ===
Return ONLY this JSON:
{
"skills": [
{
"title": "<3-5 words, generic>",
"principle": "<short sentences: ordered sub-steps + explicit what-not-to-do>",
"when_to_apply": "<concrete agent-observable trigger>"
}
// ... one skill per genuinely distinct sub-pattern
]
}
```

Figure 9: Prompt of the *Skill Distiller* for producing candidate skills.

```
You are the Skill Diagnoser. You evaluate ONE skill in an LLM agent's skill bank, based solely on the counterfactual leave-one-out
evidence below. The bank has finite retrieval capacity, and each retained skill must earn its slot — either by helping the worker
succeed, or by being repairable via rewrite.

=== Task ===
Decide one of: KEEP (the skill earns its slot as it stands); REWRITE (the skill has merit but its current wording causes harm, and a
reworded version would better serve the worker); REMOVE (the skill does not earn its slot — either it leads the worker astray, or it
is so inert that the slot would serve the worker better filled by another skill). REMOVE is irreversible in this pipeline: removed skills
cannot be auto-recovered by subsequent rounds. REMOVE should therefore have a higher evidence bar than REWRITE; between
the two, default to REWRITE unless the evidence for REMOVE is overwhelming.

=== The Skill ===
Skill ID: {skill_id}  |  Title: {title}
When to apply: {when_to_apply}
Principle:  p1. {principle_sentence_1}  p2. {principle_sentence_2}  ...

=== Evidence ({total_n} retrievals) ===
All {total_n} cases are shown below, grouped by outcome pattern. "WITH skill" means the agent ran the episode with this skill in its
retrieved set; "WITHOUT skill" means the same episode was re-run with this specific skill removed, everything else identical.

--- Group: fail→success ({n} cases; skill hurt — removing it would have succeeded) ---
[ep {episode_id}]  Task: {task_description}  |  WITH: FAIL/{n_with} steps  |  WITHOUT: SUCCESS/{n_without} steps
  WITH-skill trajectory:    1:{action_1}→{observation_1};  2:{action_2}→{observation_2};  ...
  WITHOUT-skill trajectory: 1:{cf_action_1}→{cf_observation_1};  2:{cf_action_2}→{cf_observation_2};  ...
... (one block per case)
--- Group: success→fail ({n} cases; skill helped — removing it would have failed) --- (same per-case schema)
--- Group: fail→fail ({n} cases; both failed — inherent ambiguity) --- (same per-case schema)
--- Group: success→success ({n} cases; both succeeded — efficiency signal) --- (same per-case schema)

=== Output ===
Return one JSON object:
{
  "verdict": "KEEP" | "REWRITE" | "REMOVE",
  "confidence": "high" | "medium" | "low",
  "flaw_identified": {...},
  "reasoning": "...",
  "new_title": "<full replacement title; required if title is rewritten, null otherwise>",
  "new_when_to_apply": "<full replacement when_to_apply text; required if when_to_apply is rewritten, null otherwise>",
  "new_principle": "<full replacement principle text; required if any principle sentence is rewritten, null otherwise>"
}
Judgment criteria:
  - Confidence reflects how strongly the evidence supports the verdict. Set "high" only when the harm/benefit pattern is self-
evident from consistent cases; "medium" when the pattern is discernible but has counter-examples; "low" when evidence is thin or
ambiguous.
  - flaw_identified.located_at must point to a specific sentence (pN), the title, or the when_to_apply condition — not the umbrella
label "principle". If the flaw spans multiple sentences, pick the earliest affected one.
  - new_principle is required only for REWRITE and must be the full replacement text for the entire principle (not a patch). For KEEP
and REMOVE, set new_principle to null.
  - Decide REWRITE vs REMOVE by whether harmful cases form a recognizable pattern: traceable to the same sentence and
repairable by editing → REWRITE; the core premise itself is wrong or has no salvageable value → REMOVE. Isolated harmful cases
scattered across a large neutral sample do not form a pattern.

Return only the JSON object.
```

Figure 10: Prompt of the *Skill Diagnoser*.

You are the Edit Planner for a self-evolving skill bank. At each round, you integrate the signals produced by the Skill Distiller (ADD candidates) and the Skill Diagnoser (per-skill KEEP / REWRITE / REMOVE verdicts) into a set of coherent edit proposals over the current bank. The Skill Distiller mines candidate skills from uncovered failures; the Skill Diagnoser judges each existing skill in isolation. You see the whole picture: the current bank, all candidates, and all per-skill verdicts. Your role is to compose these signals into multiple edit proposals, each aligned with one coherent editing philosophy.

=== Current Skill Bank ({n_skills} skills) ===
{skill_id}: "{title}" | principle: {principle} | when_to_apply: {when_to_apply}
... (one block per skill in the bank)

=== Skill Diagnoser Verdicts ===
[KEEP] ({n_keep} skills): {keep_skill_ids}
[REWRITE] ({n_rewrite} skills)
{rewrite_skill_id} reason: {diagnoser_reasoning} suggested rewrite: {diagnoser_suggestion} …
... (one block per REWRITE verdict)
[REMOVE] ({n_remove} skills)
{remove_skill_id} reason: {diagnoser_reasoning}
... (one block per REMOVE verdict)

=== ADD Candidate Pool ({n_candidates} items) ===
[C_0] | title: {title} | principle: {principle} | when_to_apply: {when_to_apply}
... (one block per candidate, indexed C_0 ... C_{n_candidates_max})

=== Authority ===
You operate on three pools with the following hard constraints: every ADD edit must reference a candidate_idx from the ADD pool above; every REWRITE edit must target a skill_id listed in the REWRITE pool; every REMOVE edit must target a skill_id listed in the REMOVE pool; and skills in the KEEP list are protected, so they cannot be rewritten or removed in this round.

=== Proposal Requirements ===
Generate as many distinct coherent edit proposals as the round signals support. Each proposal is a "lens" — a single editing philosophy that can be articulated in one sentence — and must: (1) be coherent (its ADD / REWRITE / REMOVE subsets serve one stated philosophy); (2) be Pareto-relevant (occupy an editing direction not already covered by another proposal); (3) be grounded (all subsets are drawn only from the three pools above); (4) be distinct from other proposals by subset composition, not merely by edit count. A proposal may act on any subset of the three pools, including only one pool when the philosophy calls for that.

=== Edit Format ===
ADD: {"type": "add", "candidate_idx": <int>} | REWRITE: {"type": "rewrite", "target_skill": "<skill_id>", "field": "principle"|"title"|"when_to_apply", "new_value": "<text>"} | REMOVE: {"type": "remove", "target_skill": "<skill_id>"}

=== Output ===
Return a JSON array of proposals. Each item has the schema:
{
"proposal_id": "<unique short identifier>",
"strategy": "<= 15 words describing the lens philosophy>",
"edits": [<edit object>, ...],
"global_observations": "<1-2 sentences explaining why this proposal is Pareto-distinct from the other proposals>"
}
Return the JSON array only.

Figure 11: Prompt of the *Edit Planner*.