

EvoSkillBank: Hierarchical Skill Self-Evolution and Skill-Bank Governance for Continual Agent Learning

Shuaiyu Zhou^{1,2,*†} Jiaying Zhang^{1,2,*†} Taoyuan Li^{3,*} Lei Shi¹
Fengpeng Yue¹ Cao Liu¹ Ke Zeng¹

¹Meituan, Beijing, China

²Peking University, Beijing, China ³Yunnan University, Kunming, China
{monster290,zhangjy2002}@stu.pku.edu.cn, litaoyuan-ujqy@stu.ynu.edu.cn
{shilei74,yuefengpeng02,liucaozengke}@meituan.com

[†] Work done during internship at Meituan. ^{*}Equal contribution.

Abstract

Agent skills are a natural substrate for continual learning in large language model agents because they allow useful procedural knowledge to be loaded only when needed rather than kept in context at all times, thereby supporting progressive disclosure of external experience and tools. However, practical skill learning still faces two coupled challenges: skills must keep evolving from execution-time experience to adapt to tasks, and, once large numbers of new skills are appended, there is no effective governance mechanism that respects the unique properties of skills as reusable procedural objects. In this paper, we introduce **EvoSkillBank**, a controlled continual skill learning framework that enables agents to self-evolve skills from benchmark interaction traces while maintaining a persistent and governed SkillBank. EvoSkillBank combines a test-time self-evolution process for turning raw trajectories into reusable skill candidates with a SkillBank evolution framework that regulates how newly generated skills are added, merged, deprecated, or rejected. Experiments on ALFWorld, WebShop, and search-augmented QA show that EvoSkillBank consistently improves over direct inference and static skill repositories, while yielding a compact and increasingly stable SkillBank under continual growth. We release the repository for both our self-evolution framework and the governed skill storage system to support future research on controlled continual skill learning for agentic LLMs.

1 Introduction

As LLM agents are increasingly deployed in persistent and open-ended settings, a central question is no longer only how to solve a task, but how to become better over time. Skills have emerged as a particularly effective substrate for continual agent learning. They support *breadth* by allowing new capabilities to be added incrementally as agents encounter new task types, and they support *depth* by enabling the refinement of existing procedures as experience accumulates on recurring problems. Compared with raw trajectories or generic memory fragments, skill is therefore a more suitable abstraction for continual learning, since it captures reusable procedural knowledge in a form that can be retrieved, improved, and transferred across tasks. Recent work on agent memory and self-evolution has increasingly moved in this direction: Evo-Memory (Wei et al., 2025) argues that agents should not merely recall past context but continuously retrieve, adapt, and evolve memory over streaming task sequences during deployment, ReasoningBank (Ouyang et al., 2026) shows that agents can distill transferable reasoning strategies from both successful and failed experiences and feed them back into future decision making, and SkillRL (Xia et al., 2026) further demonstrates that a skill library can be recursively expanded and refined alongside reinforcement learning so that skills co-evolve with the agent’s policy over training. Against this backdrop, building

a continually updated SkillBank for a target domain is becoming an important direction for enabling continual learning and self-evolution in LLM agents.

However, despite the growing recognition of skills as a more suitable carrier for continual learning, there is still no systematic strategy for the self-evolution of SkillBank (Wang et al., 2023; Xia et al., 2026). On the one hand, existing studies on skills mainly focus on skill construction, abstraction, retrieval, and invocation (Wang et al., 2023; 2025b; Yang et al., 2026), while lacking an end-to-end mechanism to continuously distill the large amount of success and failure trajectories generated at test time into accumulable, updatable, and reusable skill assets. As a result, although skills are naturally suited for continual learning, their evolution from raw experience to stable and maintainable units in SkillBank remains underexplored. On the other hand, existing experience-based agent memory methods are primarily designed to store and retrieve historical dialogues, external references, or episodic traces (Zhong et al., 2023; Packer et al., 2024; Shu et al., 2026; Yu et al., 2026), rather than task-specific strategy units with clear applicability conditions and execution principles. This mismatch becomes particularly evident for domain-specific skills: unlike conventional memory, a skill must capture not only *when to use*, but also *how to use*, and *what to do*. Therefore, directly treating skills as ordinary memory entries for appending, summarizing, or retrieval is insufficient to support their stable evolution and systematic management at the SkillBank level.

To address these challenges, we propose **EvoSkillBank**, a self-evolving SkillBank framework for continual agent learning. The key idea is to treat skill learning not as one-time accumulation, but as a closed-loop process in which execution experience is continuously transformed into reusable skills and then regulated before entering long-term storage. On the construction side, EvoSkillBank converts test-time queries, trajectories, and outcome signals into raw experience, and progressively turns them into stable skills through experience distillation, pattern aggregation, skill synthesis, and validation. On the maintenance side, EvoSkillBank introduces a governance mechanism for large-scale continual skill growth, where newly generated skills are checked against existing ones for updating, merging, deprecation, and conflict resolution before they are committed to the SkillBank. In this way, EvoSkillBank enables the SkillBank to grow over time without degenerating into an unstructured collection of redundant or conflicting entries.

In summary, our key contributions are threefold:

- We propose a hierarchical self-evolution framework that turns raw test-time experience into reusable skills for continual agents, progressively organizing skill formation through experience distillation, experience aggregation, and skill synthesis.
- We present the first governance framework for a self-evolving SkillBank. It supports incremental insertion, updating, deprecation, merging of similar skills, and conflict resolution, enabling the SkillBank to remain compact and consistent under continual growth.
- We evaluate EvoSkillBank on three representative benchmarks. The results show that the framework can effectively convert test-time experience into reusable skills, while improving both task performance and system efficiency in continual agent learning.

2 Related works

2.1 Agent skills

In recent years, research on agent skills has gradually expanded from skill construction to broader lifecycle management. Early work such as Voyager (Wang et al., 2023) explored the use of skill libraries to support skill accumulation and reuse in long-horizon tasks. Subsequent studies further investigated how reusable skills can be induced or abstracted for agentic problem solving, such as ASI (Wang et al., 2025b) and SkillWeaver (Yang et al., 2026). Beyond skill construction, recent work has also begun to address skill quality control:

SkillWeaver (Yang et al., 2026) and CRAFT (Yuan et al., 2024) explored mechanisms related to validation, specialization, and repository organization. In addition, PolySkill (Yu et al., 2025b) and WALT (Prabhu et al., 2025) studied how skills or tools can generalize across tasks, improving their reusability in broader environments. More recently, SkillRL (Xia et al., 2026) took a step toward continuous skill evolution by introducing a recursive skill-augmented reinforcement learning framework. However, existing work still mainly focuses on individual stages such as construction, validation, or generalization, rather than providing an end-to-end framework that supports skill refinement, management, and continual updating from test-time experience. To address this gap, we propose **EvoSkillBank**, which refines skills from agent experience, validates them as reusable assets, and continuously updates them to support long-term evolution.

2.2 Agent memory

In recent years, memory research in large language models (LLMs) has evolved from passive storage to active management (Hu et al., 2026). Early works like MemoryBank (Zhong et al., 2023), Mem0 (Chhikara et al., 2025), and MemGPT (Packer et al., 2024) proposed scalable memory architectures, overcoming context window limitations to enable persistent retrieval of historical information. Later, works such as MemoRAG (Qian et al., 2025) integrated retrieval-augmented generation (RAG) with memory structures to enhance complex task execution. As research progressed, the focus shifted to memory self-evolution and reasoning enhancement. ReMEM (Shu et al., 2026) introduced a two-stage framework that converts experiences into hybrid memory graphs with a temporal dimension for iterative retrieval and reasoning. MemAgent (Yu et al., 2025a) proposed a memory-based agent workflow that supports continuous execution and memory utilization, while EvoMemory (Wei et al., 2025) emphasized streaming updates and optimization of memory content. ReasoningBank (Ouyang et al., 2026) extracted transferable reasoning strategies from an agent’s successes and failures, building a memory bank to optimize task execution. As the need for more controlled memory management grew, works like Agentic Memory (Yu et al., 2026) and A-MEM (Xu et al., 2025) focused on enabling agents to autonomously manage memory storage, retrieval, and updates. Despite these advances, these works primarily enhance the ability to “remember and retrieve” past experiences, rather than distilling test-time success/failure trajectories into reusable strategy units (i.e., skills) with clear boundaries, and they do not systematically manage the full lifecycle of skills.

3 Methodology

As shown in Figure 1, **EvoSkillBank** frames continual learning around skills as the persistent unit of adaptation in a SkillBank system. It addresses two coupled challenges: transforming instance-specific test-time experience into reusable skill abstractions, and maintaining a growing SkillBank without redundancy, conflict, or retrieval degradation. To this end, it combines a hierarchical self-evolution process that induces skill candidates from accumulated experience with a governance mechanism that regulates how new skills are incorporated into the bank. Together, these components enable continual skill growth while preserving the compactness, consistency, and retrieval quality of the overall skill system.

3.1 Hierarchical self-evolution of skills

The goal of hierarchical self-evolution is to convert raw test-time trajectory experience into reusable skills. Here, “hierarchical” refers to the progressive abstraction from trajectories to experience objects, recurring patterns, and reusable skills, rather than to a tree-structured taxonomy in the SkillBank. The initial SkillBank is constructed with GPT-4o, while post-deployment self-evolution is performed by the agent backbone itself. A skill must capture a recurring behavioral regularity, expose its applicability conditions, and preserve its operational structure at a level that supports future retrieval and reuse. We provide additional details of the initial SkillBank in Appendix A. We therefore model skill formation as a progressive abstraction process that lifts execution experience from instance level to skill level through three stages: experience distillation, cross-episode aggregation, and skill synthesis.

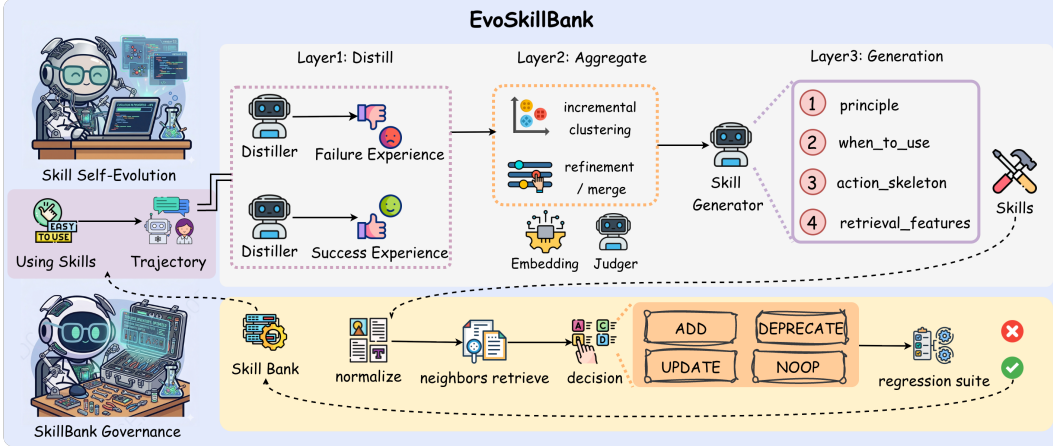


Figure 1: Overview of **EvoSkillBank**. The framework consists of two coupled components: hierarchical skill self-evolution, which distills and aggregates episodic experience into skill candidates, and SkillBank governance, which regulates promotion through neighbor retrieval, structured decision making, and regression testing.

From episodes to distilled experience. Let $\mathcal{E} = \{e_1, \dots, e_N\}$ denote the set of collected episodes. Each episode $e \in \mathcal{E}$ contains a query q , an execution trace τ , an outcome record o , auxiliary judgment artifacts j , and the set of retrieved skills r that participated in the run. Although this representation is information-rich, it is not yet suitable for long-term reuse. Many trajectory segments are instance-contingent, while the conditions under which the behavior should generalize are only implicit.

To isolate the reusable signal, we map each episode into a compact experience object

$$x = \phi(e),$$

where ϕ is a distillation procedure designed around properties that matter for skill reuse rather than trajectory reconstruction. In particular, each x preserves three kinds of information: (i) a compact description of the task situation, capturing structural properties of the query that influence strategy choice; (ii) grounded observations from the execution that explain why a strategy succeeded or failed; and (iii) reusable procedural content, expressed either as effective key actions or as failure-related constraints. Since successful and failed runs provide different supervisory signals, we maintain two experience spaces,

$$\mathcal{X}^+ = \{\phi(e) : e \text{ is successful}\}, \quad \mathcal{X}^- = \{\phi(e) : e \text{ is unsuccessful}\}.$$

The former emphasizes positive execution patterns and triggering cues, while the latter encodes breakdown modes, missing preconditions, and boundary conditions under which a strategy should not be applied.

From distilled experience to recurring patterns. A single distilled experience is still local. Skills, however, should reflect regularities that recur across multiple episodes. We therefore aggregate experience across $\mathcal{X}^+$ and $\mathcal{X}^-$ to identify stable behavioral patterns that are supported by repeated evidence. Directly clustering all structured records is brittle because irrelevant schema variation can dominate semantic similarity. Instead, for each experience object x , we construct a canonical text representation $t(x)$ that foregrounds the components most relevant to skill formation: task structure, applicability cues, decisive steps for success, and, for failed cases, characteristic failure patterns and violated assumptions. This yields a textual set

$$\mathcal{T} = \{t(x) : x \in \mathcal{X}^+ \cup \mathcal{X}^-\},$$

which serves as the semantic interface for aggregation. We then organize $\mathcal{T}$ into clusters

$$\mathcal{C} = \{C_1, \dots, C_M\}$$

using an incremental similarity-based assignment procedure. The purpose of clustering here is not merely compression but to recover candidate behavioral invariants from dispersed episodic evidence.

For each new experience text, we assign it incrementally by similarity: clearly matched cases are absorbed into existing clusters, clearly mismatched ones instantiate new clusters, and only borderline cases require semantic judgment beyond embedding geometry. Not all clusters are promoted to skills, however. We retain only mature clusters with sufficient support and coherent internal evidence, since a reusable skill should reflect a stable operational pattern rather than an accumulation of isolated anecdotes.

From mature clusters to skill candidates. Given the set of mature clusters $\mathcal{C}_{\text{mature}} \subseteq \mathcal{C}$, the final step is to synthesize each cluster into a structured skill candidate:

$$\psi : \mathcal{C}_{\text{mature}} \rightarrow \mathcal{S}_{\text{cand}}.$$

Each candidate skill $s \in \mathcal{S}_{\text{cand}}$ is not a summary of a single trajectory, but a normalized abstraction grounded in multiple related experiences. It encodes the task family, core principle, applicability conditions, action skeleton, retrieval features, and corresponding failure modes and guardrails. As in experience distillation, skill synthesis is performed through multiple constrained passes to ensure that the resulting skill remains faithful to cluster-level evidence while conforming to a reusable schema.

3.2 SkillBank governance for massive skills

Continual synthesis of new skill candidates does not necessarily improve the skill system as a whole. As the SkillBank grows, maintaining its quality becomes a challenge, since newly generated skills may overlap with existing entries, introduce semantic inconsistencies, or weaken retrieval quality if incorporated without control. For this reason, we formulate skill incorporation as a governed promotion process over an evolving repository.

Let $\mathcal{B} = \{b_1, \dots, b_L\}$ denote the current SkillBank and let $s \in \mathcal{S}_{\text{cand}}$ be a newly synthesized candidate. The governance module decides whether s should be added as a new skill, merged into an existing one, used to replace an obsolete entry, or rejected. Formally, it defines a decision rule

$$g(s, \mathcal{B}) \rightarrow a, \quad a \in \{\text{ADD}, \text{UPDATE}, \text{DEPRECATE}, \text{NOOP}\}.$$

Here, ADD inserts s as a new skill, UPDATE merges its content into an existing entry, DEPRECATE retires an older skill in favor of a stronger replacement, and NOOP discards the candidate when it is redundant, insufficiently supported, or unsafe to include.

Retrieving candidate neighbors. For each synthesized skill candidate, we retrieve a small set of relevant existing skills from the current SkillBank as candidate neighbors. This retrieval step avoids operating over the full repository and provides the local context needed for subsequent governance decisions. We retrieve a neighborhood

$$\mathcal{N}(s) \subseteq \mathcal{B}, \quad |\mathcal{N}(s)| = k,$$

Structured relation judgment. Given a candidate skill s and its retrieved neighborhood $\mathcal{N}(s)$, governance evaluates their relations through a structured comparison tailored to the functional form of skills. In particular, we compare skills along three main aspects: *when to use*, derived from applicability conditions and retrieval cues; *how to use*, derived from the action skeleton; and *what procedure to follow*, derived from the concrete operational content encoded in the skill. These structured signals determine the governance action. A candidate is ADD if it covers a previously uncovered skill region; UPDATE is triggered when it aligns with an existing skill but provides a more complete or better bounded formulation; DEPRECATE applies when it effectively supersedes an older skill; and NOOP is used when the candidate is redundant, weakly supported, or introduces harmful conflict. When these relations are ambiguous, the language model is used only for local judgment, while the final action is selected by the governance logic.

Regression test for promotion. Before committing a governance action, we evaluate the updated SkillBank on a held-out subset of benchmark queries by running inference before

Method	ALFWorld							WebShop	
	Pick	Look	Clean	Heat	Cool	Pick2	All	Score	Succ.
<i>Base Models</i>									
Qwen2.5-7B-Instruct	34.2	22.1	20.0	8.0	3.3	4.2	15.3	27.2	7.4
<i>Prompt-based or Memory-based Methods</i>									
ReAct	44.7	32.8	31.5	12.1	16.9	15.3	25.6	43.8	17.6
Reflexion	58.4	39.7	41.6	28.5	33.1	<u>21.2</u>	37.1	<u>54.6</u>	<u>25.9</u>
Mem0	50.8	51.6	24.1	33.2	18.7	6.5	30.8	22.4	1.8
ExpeL	19.6	<u>62.1</u>	51.4	<u>48.8</u>	64.2	5.2	41.9	28.7	9.6
MemP	51.2	36.7	45.3	52.7	29.4	15.1	38.4	23.9	5.7
SimpleMem	<u>60.6</u>	31.4	18.2	11.4	30.6	3.2	25.9	31.1	7.8
Voyager	58.9	57.6	47.3	<u>48.8</u>	25.4	15.2	<u>42.2</u>	47.8	24.3
EvoSkillBank	72.8	64.9	<u>47.6</u>	39.8	<u>35.4</u>	30.1	48.4	61.7	31.5

Table 1: Performance on ALFWorld and WebShop under a unified evaluation setting. Bold and underline indicate the best and second-best results.

and after the proposed update. Let $\mathcal{Q}_{\text{eval}}$ denote the evaluation subset. For each $q \in \mathcal{Q}_{\text{eval}}$, we execute the agent with the pre-update and post-update SkillBank and compare the resulting performance. A proposed update is accepted if it yields improvement, or if any degradation remains within a predefined tolerance. Otherwise, the SkillBank is rolled back to its previous state and the proposed evolution is discarded.

Overall, EvoSkillBank closes the loop of continual SkillBank evolution by coupling a three-level hierarchical self-evolution process with governed skill incorporation. In this way, skill growth is not treated as unconstrained accumulation, but as a controlled closed-loop process that jointly promotes expansion, consistency, and retrieval quality of the SkillBank over time. The detailed prompts used for both skill evolution and governance are provided in Appendix C. In addition, to further illustrate how skills function during inference and execution, we present a case study in Appendix D.

4 Experiments

4.1 Experimental setup

Benchmarks. We evaluate EvoSkillBank on three benchmark families covering complementary forms of agentic behavior. ALFWorld (Shridhar et al., 2021) measures long-horizon embodied decision making, where we report success rates on its six standard household subtasks and their overall average. WebShop (Yao et al., 2023a) evaluates multi-step goal-conditioned web interaction in a simulated e-commerce environment, and we report both average score and task success rate. In addition, we evaluate search-augmented question answering on single-hop datasets (NQ (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), and PopQA (Mallen et al., 2023)) and multi-hop datasets (HotpotQA (Yang et al., 2018), 2Wiki (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023)). This setup tests continual skill learning across embodied, interactive, and retrieval-intensive settings.

Baselines. We compare EvoSkillBank against representative baselines from the main competing paradigms. For ALFWorld and WebShop, we reproduce ReAct (Yao et al., 2023b), Reflexion (Shinn et al., 2023), Mem0 (Chhikara et al., 2025), ExpeL (Zhao et al., 2024), MemP (Fang et al., 2026), SimpleMem (Liu et al., 2026), and Voyager (Wang et al., 2023) under the same evaluation setting. We do not include SkillRL and Skill0 (Xia et al., 2026; Lu et al., 2026) in the main comparison because they rely on training-based policy or skill optimization, whereas EvoSkillBank is a training-free test-time method. For search-augmented QA, we compare against Search-o1 (Li et al., 2025), Search-R1 (Jin et al., 2025), ZeroSearch (Sun et al., 2025), and StepSearch (Wang et al., 2025a), covering standard prompt-

Method	Single-Hop QA			Multi-Hop QA				Avg.
	NQ [†]	TriviaQA‡	PopQA‡	HotpotQA [†]	2Wiki‡	MuSiQue‡	Bamboogle‡	
<i>Base Models</i>								
Qwen2.5	11.6	35.6	1.2	16.4	22.2	4.8	14.4	15.2
<i>Prompt-based Methods</i>								
CoT*	12.8	35.6	3.8	16.2	22.6	6.6	24.0	17.4
RAG*	27.4	58.2	17.8	25.8	23.2	9.4	16.8	25.5
Search-o1*	19.4	40.6	11.4	17.0	27.0	8.6	30.4	22.1
R1-Instruct*	21.0	44.9	17.1	20.8	27.5	6.0	19.2	22.4
Search-R1*	39.3	61.0	39.7	37.0	40.1	14.6	36.8	38.5
ZeroSearch*	<u>43.6</u>	61.8	51.5	34.6	35.2	18.4	27.8	39.1
StepSearch*	–	–	–	<u>38.6</u>	36.6	22.6	<u>40.0</u>	–
EvoSkillBank	44.5	62.3	<u>47.0</u>	39.5	41.0	<u>18.8</u>	41.1	42.0

Table 2: Performance on search-augmented QA tasks. We compare EvoSkillBank against representative prompt-based baselines. [†] indicates in-domain datasets, [‡] indicates out-of-domain datasets, and * indicates results replicated from (Xia et al., 2026).

ing, retrieval-augmented generation, and search-based reasoning. We additionally report the corresponding Qwen2.5-7B-Instruct (Bai et al., 2023) base model on each benchmark family as a shared backbone reference. A comparison with ReasoningBank is provided in Appendix B.

Evaluation protocol. We follow the verl-agent (Feng et al., 2025) evaluation framework for benchmark inference. After each round, the collected trajectories are fed into EvoSkillBank for hierarchical self-evolution and SkillBank governance, yielding an updated SkillBank for the next round. No model parameters are updated; adaptation is achieved solely through controlled SkillBank evolution. For retrieval, we use Qwen3-Embedding-0.6B (Zhang et al., 2025), store the SkillBank in SQLite (sql), and retrieve the top-3 skills. The overall inference pipeline is kept aligned with SkillRL (Xia et al., 2026).

4.2 Main results

Tables 1 and 2 summarize the main results. Overall, EvoSkillBank consistently improves over the base model across all three benchmark families, showing that training-free continual skill evolution is effective for enhancing agent performance.

ALFWorld and WebShop. On ALFWorld and WebShop, EvoSkillBank achieves the best overall performance under the unified evaluation setting. It reaches 48.4% on ALFWorld, outperforming the strongest baseline, Voyager, by 6.2 points. On WebShop, it obtains a score of 61.7 and a success rate of 31.5%, improving over the strongest corresponding baselines by 7.1 and 5.6 points, respectively.

Search-augmented QA. On search-augmented QA, EvoSkillBank also achieves the best overall average result, reaching 42.0. The results indicate that the evolved SkillBank improves both evidence retrieval and multi-step reasoning, leading to consistent gains on both single-hop and multi-hop QA benchmarks.

4.3 Ablation study

We conduct an additive ablation study on Qwen2.5-7B-Instruct to isolate the contribution of each component in EvoSkillBank. Starting from the Base Model, we progressively add each module to form the full system, so that performance changes directly reflect the incremental effect of each component. Self-Evolution introduces hierarchical self-evolution, distillation, and experience aggregation, with generated skills directly added to the SkillBank. Governance (w/o RegTest) further adds neighbor retrieval and structured promotion decisions, but without final regression validation. Regression Test (Full) completes the framework by validating each proposed SkillBank update on a held-out subset before commitment.

Method	ALFWorld	WebShop	Hop QA
Base Model	15.3	27.2	15.2
+ Seed Bank(with 156-Skill)	28.3	43.7	24.3
+ Self-Evolution	41.8	56.8	39.4
+ Governance (w/o RegTest)	46.4	60.1	34.3
+ Regression Test (Full)	48.4	61.7	42.0

Table 3: Additive ablation of the seed bank, self-evolution, governance, and regression testing.

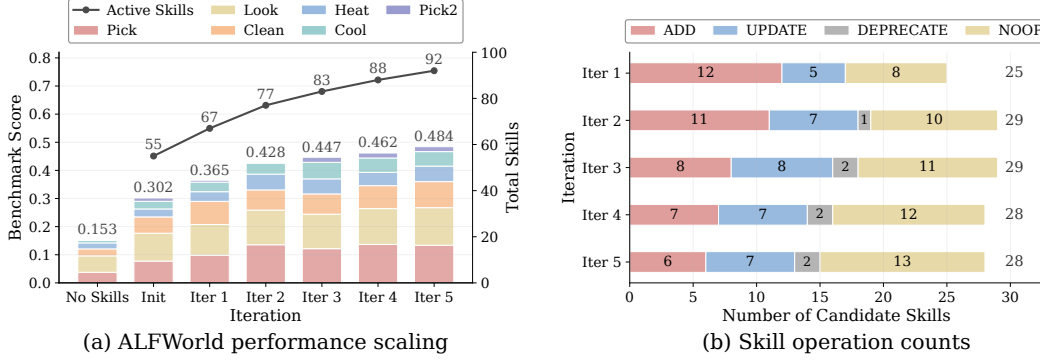


Figure 2: Evolution dynamics of EvoSkillBank. (a) ALFWorld performance improves rapidly at first and then gradually saturates, while active skills grow more slowly. (b) Governance outcomes shift from frequent additions to more selective updates and no-op decisions.

Table 3 separates the contribution of the 156-skill seed bank from subsequent SkillBank evolution. The seed bank improves all three benchmark families, and self-evolution provides further gains. Governance without regression testing improves ALFWorld and WebShop but reduces Hop QA from 39.4 to 34.3 because locally plausible updates can still harm fine-grained reasoning skills. Regression testing filters such updates and raises the full system to 48.4, 61.7, and 42.0 on ALFWorld, WebShop, and Hop QA, respectively.

5 Analysis

We provide a more detailed analysis of EvoSkillBank in the ALFWorld environment beyond the final results. Specifically, we examine how performance evolves across continual iterations, how the SkillBank grows over time, and how governance shapes the evolving repository.

5.1 Evolution dynamics

Figure 2(a) shows the performance trajectory of EvoSkillBank over continual iterations on ALFWorld. Performance improves substantially in the early rounds and then gradually approaches saturation, indicating that high-value reusable skills are acquired early, while later rounds mainly contribute smaller refinements. This trend is consistent with the objective of skill evolution: the system first captures the most frequent and transferable behaviors, and then shifts toward incremental improvement.

The same figure also shows that gains are accompanied by a steady increase in the number of active skills. Importantly, performance does not improve only because the repository becomes larger; rather, the largest gains occur when the skill set is still relatively compact, suggesting that early-added skills carry disproportionate utility.

5.2 SkillBank growth

Figure 2 further characterizes how the SkillBank evolves over time. In Figure 2(a), the number of active skills increases quickly in the first few rounds and then slows down, indicating that EvoSkillBank progressively covers the most reusable patterns instead of expanding without bound. In parallel, the benchmark score continues to improve even as growth decelerates, suggesting a transition from rapid expansion to more selective refinement.

Figure 2(b) provides a more detailed view of this process by showing the governance outcomes for candidate skills at each iteration. Early rounds contain more ADD operations, while later rounds exhibit fewer accepted additions and more NOOP decisions. UPDATE and DEPRECATE operations remain consistently present, showing that later-stage evolution increasingly revises and consolidates existing skills rather than simply appending new ones. Together, these patterns indicate that the SkillBank moves toward a more stable and compact repository over time.

5.3 Retrieval quality and governance

We evaluate retrieval quality on the top- k retrieved skills for each query, with $k = 5$ in our experiments. For an evaluation query $q \in \mathcal{Q}_{\text{eval}}$, let the retrieved set be $R_k(q) = \{s_1, \dots, s_k\}$. The LLM judgments for **SkillMatch**, **ConflictRate**, and **OverlapRate** use the prompts in Appendix C (Prompts P, Q, and R). As shown in Figure 3, governance becomes increasingly important as the SkillBank grows: although both variants benefit from self-evolution in early rounds, the full EvoSkillBank pipeline achieves higher SkillMatch and lower ConflictRate and OverlapRate in later iterations, indicating a cleaner and more query-appropriate retrieval set over time.

SkillMatch measures whether the retrieved skills are individually useful for the query. With binary labels $m_i(q) \in \{0, 1\}$, we define

$$\text{SkillMatch} = \frac{1}{|\mathcal{Q}_{\text{eval}}|} \sum_{q \in \mathcal{Q}_{\text{eval}}} \left(\frac{1}{k} \sum_{i=1}^k m_i(q) \right).$$

ConflictRate measures whether the retrieved set contains mutually conflicting skills. With pairwise conflict labels $c_{ij}(q) \in \{0, 1\}$ over the $\binom{k}{2}$ skill pairs, we define

$$\text{ConflictRate} = \frac{1}{|\mathcal{Q}_{\text{eval}}|} \sum_{q \in \mathcal{Q}_{\text{eval}}} \left(\frac{1}{\binom{k}{2}} \sum_{1 \leq i < j \leq k} c_{ij}(q) \right).$$

OverlapRate is defined analogously using pairwise overlap labels $o_{ij}(q) \in \{0, 1\}$:

$$\text{OverlapRate} = \frac{1}{|\mathcal{Q}_{\text{eval}}|} \sum_{q \in \mathcal{Q}_{\text{eval}}} \left(\frac{1}{\binom{k}{2}} \sum_{1 \leq i < j \leq k} o_{ij}(q) \right).$$

Benchmark	ADD	UPDATE	DEPRECATE	NOOP	Accuracy
ALFWorld	61/63	54/52	27/26	58/59	93.5%
WebShop	48/50	67/65	24/23	61/62	93.0%
QA tasks	42/44	71/68	18/19	69/69	92.5%

Table 4: Human–LLM agreement on SkillBank governance decisions.

Additional evaluations of governance reliability, generalization across backbones and environments, and long-horizon SkillBank maintenance are provided in Appendix B.

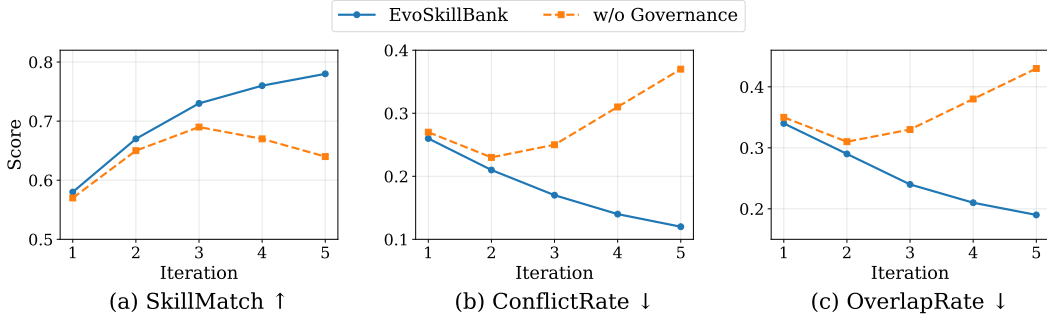


Figure 3: Retrieval quality over continual iterations. Governance improves the usefulness of the top-5 retrieved skills while reducing conflict and redundancy in the retrieved skill set.

Harmful update type	Without RegTest	Rolled back
Over-generalized UPDATE	17	14
Noisy ADD	13	11
Incorrect DEPRECATE	9	8
Retrieval-conflict increase	15	13
Total	54	46

Table 5: Harmful Hop QA updates and the operations rejected by regression testing.

5.4 Governance reliability and regression safeguards

To validate governance decisions, we randomly sample 200 candidate experiences from each benchmark family before SkillBank promotion and compare the LLM decision with human annotation. Each entry in Table 4 reports the Human/LLM counts for an operation.

The decisions achieve 92.5–93.5% agreement. Most disagreements occur in ambiguous boundary cases, especially between UPDATE and NOOP, or between UPDATE and DEPRECATE.

We further inspect harmful governance operations accepted without regression testing on Hop QA. As shown in Table 5, most harmful cases are over-generalized updates, noisy additions, incorrect deprecations, or operations that increase retrieval conflict.

Regression testing rolls back 46 of the 54 harmful updates. This explains why governance without RegTest reduces Hop QA performance, while the full system restores it from 34.3 to 42.0.

6 Conclusion

In this paper, we presented **EvoSkillBank**, a controlled continual skill learning framework for agentic large language models. Starting from raw benchmark interaction traces, EvoSkillBank progressively transforms execution-time experience into reusable skill candidates and further regulates how those skills are incorporated into an evolving SkillBank. Unlike prior approaches that treat skill accumulation as a largely append-only process, EvoSkillBank frames continual skill learning as a closed loop of self-evolution and repository governance. Empirically, we showed that EvoSkillBank consistently improves over both direct inference and static skill repositories. Our current framework opens several directions for future work. One natural extension is to study richer forms of skill composition and hierarchical reuse across tasks. Another is to investigate stronger evaluation protocols for long-horizon repository evolution, including larger-scale multi-domain settings and longer continual learning horizons.

Acknowledgments

This work was supported by the Beijing Nova Program. We thank our collaborators and institutions for providing computational resources, API access, engineering support, and valuable discussions on algorithm development. We also thank the reviewers and program chairs for their constructive feedback.

References

- Sqlite. URL <https://sqlite.org/about.html>.
- Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report, 2023. URL <https://arxiv.org/abs/2309.16609>.
- Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL <https://arxiv.org/abs/2504.19413>.
- Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory, 2026. URL <https://arxiv.org/abs/2508.06433>.
- Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.
- Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In Donia Scott, Nuria Bel, and Chengqing Zong (eds.), *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.580. URL <https://aclanthology.org/2020.coling-main.580/>.
- Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of ai agents, 2026. URL <https://arxiv.org/abs/2512.13564>.
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Serkan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning, 2025. URL <https://arxiv.org/abs/2503.09516>.
- Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017. URL <https://arxiv.org/abs/1705.03551>.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl.a.00276. URL <https://aclanthology.org/Q19-1026/>.

- Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models, 2025. URL <https://arxiv.org/abs/2501.05366>.
- Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for llm agents, 2026. URL <https://arxiv.org/abs/2601.02553>.
- Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Skill0: In-context agentic reinforcement learning for skill internalization, 2026. URL <https://arxiv.org/abs/2604.02268>.
- Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories, 2023. URL <https://arxiv.org/abs/2212.10511>.
- 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, 2026. URL <https://arxiv.org/abs/2509.25140>.
- Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL <https://arxiv.org/abs/2310.08560>.
- Viraj Prabhu, Yutong Dai, Matthew Fernandez, Jing Gu, Krithika Ramakrishnan, Yanqi Luo, Silvio Savarese, Caiming Xiong, Junnan Li, Zeyuan Chen, and Ran Xu. Walt: Web agents that learn tools, 2025. URL <https://arxiv.org/abs/2510.01524>.
- Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models, 2023. URL <https://arxiv.org/abs/2210.03350>.
- Hongjin Qian, Zheng Liu, Peitian Zhang, Kelong Mao, Defu Lian, Zhicheng Dou, and Tiejun Huang. Memorag: Boosting long context processing with global memory-enhanced retrieval augmentation, 2025. URL <https://arxiv.org/abs/2409.05591>.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL <https://arxiv.org/abs/2307.16789>.
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL <https://arxiv.org/abs/2303.11366>.
- Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning, 2021. URL <https://arxiv.org/abs/2010.03768>.
- Yiheng Shu, Saisri Padmaja Jonnalagedda, Xiang Gao, Bernal Jiménez Gutiérrez, Weijian Qi, Kamalika Das, Huan Sun, and Yu Su. Remem: Reasoning with episodic memory in language agent, 2026. URL <https://arxiv.org/abs/2602.13530>.
- Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. Zerossearch: Incentivize the search capability of llms without searching, 2025. URL <https://arxiv.org/abs/2505.04588>.
- Harsh Trivedi, Niranjana Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition, 2022. URL <https://arxiv.org/abs/2108.00573>.

- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL <https://arxiv.org/abs/2305.16291>.
- Ziliang Wang, Xuhui Zheng, Kang An, Cijun Ouyang, Jialu Cai, Yuhang Wang, and Yichao Wu. Stepsearch: Igniting llms search ability via step-wise proximal policy optimization, 2025a. URL <https://arxiv.org/abs/2505.15107>.
- Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks, 2025b. URL <https://arxiv.org/abs/2504.06821>.
- Tianxin Wei, Naveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory, 2025. URL <https://arxiv.org/abs/2511.20857>.
- 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, 2026. URL <https://arxiv.org/abs/2602.08234>.
- Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL <https://arxiv.org/abs/2502.12110>.
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun’ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October–November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL <https://aclanthology.org/D18-1259/>.
- Ziyi Yang, Benned Hedegaard, Ahmed Jaafar, Yichen Wei, Skye Thompson, Shreyas S. Raman, Haotian Fu, Stefanie Tellex, George Konidakis, David Paulius, and Naman Shah. Skillwrapper: Generative predicate invention for task-level planning, 2026. URL <https://arxiv.org/abs/2511.18203>.
- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023a. URL <https://arxiv.org/abs/2207.01206>.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023b. URL <https://arxiv.org/abs/2210.03629>.
- Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiyang Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent, 2025a. URL <https://arxiv.org/abs/2507.02259>.
- Simon Yu, Gang Li, Weiyan Shi, and Peng Qi. Polyskill: Learning generalizable skills through polymorphic abstraction, 2025b. URL <https://arxiv.org/abs/2510.15863>.
- Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents, 2026. URL <https://arxiv.org/abs/2601.01885>.
- Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R. Fung, Hao Peng, and Heng Ji. Craft: Customizing llms by creating and retrieving from specialized toolsets, 2024. URL <https://arxiv.org/abs/2309.17428>.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models, 2025. URL <https://arxiv.org/abs/2506.05176>.

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

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory, 2023. URL <https://arxiv.org/abs/2305.10250>.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL <https://arxiv.org/abs/2307.13854>.

A Skills schema in the self-evolution process

The initial SkillBank is organized as a structured repository of reusable skills rather than a collection of raw prompts or trajectory fragments. Each skill encodes not only a strategy principle, but also its applicability conditions, procedural content, retrieval features, and operational boundaries. Because this initial repository serves as the substrate for subsequent self-evolution and governance, it is important to characterize its structure and coverage.

To make this initialization concrete, we summarize the SkillBank from three complementary perspectives. Table 6 presents the schema of an individual skill item, Table 7 and Table 8 describe the scale and composition of the library, and Table 9 shows representative examples in compressed form. Together, these views illustrate that the initial SkillBank is both structured and functionally diverse, providing a practical starting point for continual skill evolution.

Field	Description
title	Short reusable skill name
principle	Core strategy or prevention principle
when_to_use	Applicability cues and boundary conditions
action_skeleton	Reusable high-level procedure
retrieval_features	Keywords and tags for retrieval
known_failure_modes	Common misuse or failure patterns
guardrails	Operational constraints for safe application

Table 6: Structured schema of a skill item in the initial SkillBank.

Statistic	Value
Total number of initial skills	156
Avg. tokens per skill	118
Task families	12
Planning-stage skills	41
Execution-stage skills	73
Verification-stage skills	42
Avg. action steps per skill	3.8
Avg. guardrails per skill	2.4

Table 7: Summary statistics of the initial SkillBank.

Skill category	Count
Fact collection and decomposition	32
Comparison and composition	27
Disambiguation and attribute grounding	21
Verification and consistency checking	25
Failure prevention and boundary control	29
Tool-use and retrieval routines	22

Table 8: Coarse-grained categorization of initial skills.

Overall, the initial SkillBank provides a structured and diverse starting point for continual skill evolution.

B Additional experiments

B.1 Comparison with ReasoningBank

ReasoningBank (Ouyang et al., 2026) is a relevant baseline because it extracts reusable reasoning strategies from successful and failed experiences. Table 10 compares generic

Representative Skill 1 Skill ID: aleworld.004 Title: Track Counts and Progress Task family: aleworld Principle: Maintain an internal counter of how many goal objects are still needed; stop searching or terminate the episode only when the counter reaches zero. When to use: Use for tasks requiring multiple objects or multiple successful placements. Positive cues include goals requiring two or more objects, repeated instances of the same object type, and tasks where placed count matters. Action skeleton: initialize required count; update count on pickup and placement; compare placed count against goal; stop only when count is satisfied. Retrieval features: count tracking, multi-instance, progress; aleworld; household, bookkeeping. Failure modes: keeps searching after enough objects are placed; terminates before all instances are placed. Guardrail: never terminate based on intuition; use explicit counters.
Representative Skill 2 Skill ID: aleworld.032 Title: No Appliance Before Object Task family: aleworld Principle: Do not move to or interact with the microwave or final destination until the required object has been identified and picked up. When to use: Use in heat tasks whenever navigation planning is considering the microwave or destination before the target object is secured. Positive cues include knowing the microwave location early while the target object is not yet held. Action skeleton: check whether the target object is held; if not held continue object search; defer microwave and destination navigation; switch to appliance interaction only after pickup. Retrieval features: object before appliance, no microwave before pickup, search first; aleworld; household, heat, planning. Failure modes: wastes steps visiting the microwave before finding the object; oscillates between appliance and search areas. Guardrail: appliance interaction is invalid until the correct target object has been secured.
Representative Skill 3 Skill ID: aleworld.044 Title: Lock In Observed Items Task family: aleworld Principle: After spotting either a required target object or a desk lamp, act before leaving the area by picking up the target or memorizing the lamp location to avoid costly backtracking. When to use: Use in examine tasks immediately after one of the required objects is observed during search. Positive cues include newly observing a target object or desk lamp when the agent is about to move away from the area. Action skeleton: detect that a required object has been observed; if it is the target pick it up; if it is the desk lamp store its location; update the plan using this new evidence before moving on. Retrieval features: lock in observation, memorize lamp, pickup target on sight; aleworld; household, examine, memory. Failure modes: notices a useful object and leaves without using the information; has to rediscover the same target or tool later. Guardrail: do not abandon a confirmed task-relevant observation without either acting on it or recording it in memory.

Table 9: Representative skills from the initial SkillBank in compressed form.

reasoning-memory retrieval with hierarchical skill self-evolution and the full governed SkillBank.

EvoSkillBank outperforms ReasoningBank on all three benchmark families. Self-evolution also improves over ReasoningBank, while SkillBank governance provides further gains.

Method	ALFWorld	WebShop	QA tasks
ReasoningBank	37.6	53.2	35.8
Self-Evolution	41.8	56.8	39.4
EvoSkillBank	48.4	61.7	42.0

Table 10: Comparison with ReasoningBank on the three benchmark families.

B.2 Generalization across backbones and environments

We evaluate four additional backbones: Qwen3-4B-Instruct-2507, Qwen3-30B-A3B-Instruct-2507, GPT-4o, and Gemini-2.5-Pro. ReAct and EvoSkillBank use the same setting for each backbone. EvoSkillBank consistently improves over ReAct on the aggregate ALFWorld and WebShop metrics across all tested backbones, showing that the gains are not tied to Qwen2.5-7B-Instruct. We also evaluate EvoSkillBank on WebArena (Zhou et al., 2024) and ToolBench (Qin et al., 2023), which provide broader web-interaction and tool-use environments.

Model	ALFWorld							WebShop	
	Pick	Look	Clean	Heat	Cool	Pick2	Avg.	Score	Succ.
Qwen2.5-7B	44.7	32.8	31.5	12.1	16.9	15.3	25.6	43.8	17.6
Qwen3-4B	67.4	52.8	18.7	36.5	22.1	27.6	37.5	47.6	15.9
Qwen3-30B-A3B	79.6	58.4	39.7	49.1	31.8	61.2	53.3	42.5	13.2
GPT-4o	78.9	60.2	22.4	53.7	19.6	38.8	45.6	31.8	25.9
Gemini-2.5-Pro	82.6	50.9	63.8	65.1	27.4	56.7	57.8	50.4	36.5

Table 11: ReAct performance across model backbones. The first seven metrics are for ALFWorld; Score and Succ. are for WebShop.

Model	ALFWorld							WebShop	
	Pick	Look	Clean	Heat	Cool	Pick2	Avg.	Score	Succ.
Qwen2.5-7B	72.8	64.9	47.6	39.8	35.4	30.1	48.4	61.7	31.5
Qwen3-4B	75.9	58.7	51.6	53.9	63.8	38.4	57.1	50.2	35.7
Qwen3-30B-A3B	80.7	75.6	77.9	80.8	59.6	55.9	71.8	57.2	54.9
GPT-4o	88.4	86.7	84.9	88.6	78.9	85.4	85.5	73.8	63.9
Gemini-2.5-Pro	92.7	91.8	88.6	84.9	74.8	91.9	87.5	80.9	70.8

Table 12: EvoSkillBank performance across model backbones. The first seven metrics are for ALFWorld; Score and Succ. are for WebShop.

EvoSkillBank outperforms both the base model and Reflexion on WebArena and ToolBench, indicating that the SkillBank evolution mechanism also transfers to broader task distributions.

B.3 Long-horizon SkillBank maintenance

We extend the ALFWorld evolution analysis from iteration 5 to iteration 15 to evaluate longer-term SkillBank behavior.

Performance remains stable after iteration 5 while the SkillBank grows slowly from 92 to 100 active skills. This supports controlled continual refinement and stable repository maintenance rather than unbounded capability growth.

Method	WebArena success	ToolBench pass@1
Qwen2.5-7B-Instruct	8.6	44.6
Reflexion	16.4	57.1
EvoSkillBank	26.3	68.2

Table 13: Generalization to WebArena and ToolBench.

Iteration	ALFWorld	Skills in SkillBank
5	48.4	92
7	48.1	94
9	48.4	96
11	48.3	97
13	48.0	99
15	48.2	100

Table 14: ALFWorld performance and SkillBank size over extended evolution.

C Prompt templates

Prompt A: Distill Success Situation

You are distilling one successful episode into reusable EvoSkill experience.
Your task in this pass is only to identify the **situation signature** of the episode.
You should infer: - **query_shape**: short reusable labels describing the task shape - **requires_intermediate_facts**: whether the task requires collecting intermediate facts before answering - **ambiguity_present**: whether the episode contains attribute ambiguity, entity ambiguity, or concept ambiguity
Do not retell the episode. Do not describe success or failure. Only output the situation signature.
Output example:
{ "situation_signature": { "query_shape": ["comparison", "multi_hop"], "requires_intermediate_facts": false, "ambiguity_present": false } }
Use the following episode: {episode.text}

Prompt B: Distill Success Observations and Cues

You are distilling one successful episode into reusable EvoSkill experience.
Your task in this pass is to extract: 1. **supporting_observations** 2. **when_to_use_cues**
Meaning of **supporting_observations**: - short evidence-grounded observations taken from the trajectory, outcome, or judge result - these should explain why the strategy worked in this episode - do not write abstract slogans
Meaning of **when_to_use_cues**: - short reusable cues that tell the system when this kind of experience should be retrieved again - these should describe task conditions, not episode-specific names or strings - make them general enough for reuse, but not vague
Output example:
{ "supporting_observations": ["the agent collected the needed facts before answering", "the final answer was based on explicitly retrieved evidence"], "when_to_use_cues": ["use when the task requires comparing two separately retrieved values", "use when the answer depends on intermediate fact collection"] }
Use the following episode: {episode.text}

Prompt C: Distill Success Key Steps

You are distilling one successful episode into reusable EvoSkill experience.
Your task in this pass is to extract the **key reusable strategy steps**.
These steps should be: - high-level - reusable in future similar tasks - not a literal replay of the full trajectory - not specific to the current entity names
Focus only on the steps that were important to success.
Output example:
{ "key_steps": ["identify the target entities and attributes", "retrieve the necessary facts separately", "compose the answer only after evidence is complete"] }
Use the following episode: {episode_text}

Prompt D: Distill Failure Situation

You are distilling one failed episode into reusable EvoSkill experience.
Your task in this pass is only to identify the **situation signature** of the episode.
You should infer: - **query_shape**: short reusable labels describing the task shape - **requires_intermediate_facts**: whether the task requires collecting intermediate facts before answering - **ambiguity_present**: whether the episode contains attribute ambiguity, entity ambiguity, or concept ambiguity
Do not explain the failure yet. Only output the situation signature.
Output example:
{ "situation_signature": { "query_shape": ["comparison", "multi_hop"], "requires_intermediate_facts": true, "ambiguity_present": true } }
Use the following episode: {episode_text}

Prompt E: Distill Failure Observations and Failure Step

You are distilling one failed episode into reusable EvoSkill experience.
Your task in this pass is to identify: 1. **supporting_observations** 2. **failure_step** 3. **observed_failures**
Meaning of **supporting_observations**: - short direct observations grounded in the trajectory, outcome, or judge artifacts - these are pieces of evidence, not explanations
Meaning of **failure_step**: - the single most responsible stage where the failure occurred - choose a short label such as: planning, retrieval, evidence.collection, execution, verification, answer_synthesis, unknown
Meaning of **observed_failures**: - concrete failure behaviors shown in this episode - for example: collected the wrong attribute, compared too early, used a proxy attribute, skipped verification
Output example:
{ "supporting_observations": ["the trajectory searched for birthplace instead of nationality", "the judge artifact states that birthplace was used as a proxy"], "failure_step": "evidence_collection", "observed_failures": ["collected the wrong attribute type", "used birthplace as a proxy for nationality"] }
Use the following episode: {episode_text}

Prompt F: Distill Failure Boundaries

You are distilling one failed episode into reusable EvoSkill experience.
Your task in this pass is to identify: 1. **missing_preconditions** 2. **when_to_use_violation**
Meaning of **missing_preconditions**: - what should have been true before this strategy could be safely or correctly applied - examples: the exact target attribute must be identified first, both facts must be collected first, disambiguation must happen first
Meaning of **when_to_use_violation**: - how this episode reveals that the strategy was applied outside its safe boundary - this should describe when the strategy should not be used or should be constrained more tightly

Output example:

```
{ "missing_preconditions": [ "the exact target attribute must be identified before retrieval", "the compared values must correspond to the queried attribute" ],
  "when_to_use_violation": [ "do not use this strategy when the attribute type is still ambiguous", "do not compare proxy attributes as if they were the requested attribute" ] }
```

Use the following episode: {episode_text}

Prompt G: Cluster Assignment

You are deciding cluster assignment for one new experience.

A new experience is given below. There are also several candidate clusters, each represented by a few representative texts.

Your task: - If the new experience clearly belongs to one existing cluster, return that **cluster_id** - Otherwise return **"new_cluster"**

Return one JSON object only.

Example:

```
{ "decision": "success_cluster.0003" }
```

or

```
{ "decision": "new_cluster" }
```

New experience: {new_text}

Candidate clusters: {candidates_block}

Prompt H: Cluster Merge

You are deciding whether two experience clusters should be merged.

You are given two clusters represented by their representative texts.

Merge them only if they clearly describe the same reusable experience pattern. Keep them separate if they differ in task condition, strategy, failure mode, or boundary.

Return one JSON object only.

Example: {"decision": "merge"} or {"decision": "separate"}

Cluster A: {cluster_a_text}

Cluster B: {cluster_b_text}

Prompt I: Skill Basic Fields

You are generating a final SkillBank item from one experience cluster.

{cluster_role_hint}

In this pass, output **only** these 3 fields: - **task_family** - **title** - **principle**

Field requirements:

1) **title** - Short, reusable skill name - Not a long sentence - No episode-specific entity names, website names, or raw strings

2) **principle** - 1–2 sentences describing the core principle - For success clusters: summarize the winning strategy - For failure clusters: summarize the prevention principle

Return **one JSON object only**, exactly in this shape:

```
{ "title": "Collect Exact Attributes Before Comparison", "principle": "Retrieve the exact target attribute explicitly before comparing, and avoid proxy attributes unless equivalence is stated." }
```

Cluster data: {cluster_text}

Prompt J: Skill When-to-Use

You are generating the when_to_use field for a SkillBank item, using the same cluster.

Output **only**: - **when_to_use.summary** - **when_to_use.positive_cues**
 - **when_to_use.negative_cues** - **when_to_use.required_conditions** -
when_to_use.forbidden_conditions - **when_to_use.stage**

Field requirements:

- 1) **summary** - One sentence describing when this skill should be used
- 2) **positive_cues** - Concrete triggers or signals - Generalizable, not episode-specific
- 3) **negative_cues** - When this skill should not be used - Avoid vague statements
- 4) **required_conditions** - Preconditions that must hold before applying the skill
- 5) **forbidden_conditions** - Explicit situations where the skill must not be used - Can be empty
- 6) **stage** - Must be one of: planning / execution / verification

Return **one JSON object only**, exactly in this shape:

```
{ "when_to_use": { "summary": "Use when ...", "positive_cues": ["..."], "negative_cues": ["..."], "required_conditions": ["..."], "forbidden_conditions": ["..."], "stage": "planning" } }
```

Current skill fields: {current_skill_json}

Cluster data: {cluster_text}

Prompt K: Skill Structure and Retrieval

You are generating execution and retrieval fields for a SkillBank item, using the same cluster.

Output **only**: - **action_skeleton** - **retrieval_features**

Field requirements:

- 1) **action_skeleton** - 2 to 6 high-level reusable steps - Not a literal replay of any single episode
- 2) **retrieval_features** - **keywords**: short keywords for retrieval - **benchmark_tags**: include only if confidently inferable, otherwise [] - **env_tags**: include only if confidently inferable, otherwise []

Return **one JSON object only**, exactly in this shape:

```
{ "action_skeleton": [ "identify the target entities and attributes", "retrieve the required facts separately", "compose the answer only after evidence is complete" ], "retrieval_features": { "keywords": ["compare", "exact attribute", "evidence"], "benchmark_tags": ["hotpotqa"], "env_tags": ["search_agent"] } }
```

Current skill fields: {current_skill_json}

Cluster data: {cluster_text}

Prompt L: Skill Safety Boundaries

You are generating safety boundary fields for a SkillBank item, using the same cluster.

Output **only**: - **known_failure_modes** - **guardrails**

Field requirements:

- 1) **known_failure_modes** - Common ways this skill is misused or fails - For failure clusters: summarize recurring failure patterns - Keep them short and actionable
- 2) **guardrails** - Concrete constraints or checks that must be followed when executing this skill - Must be operational, not vague

Return **one JSON object only**, exactly in this shape:

```
{ "known_failure_modes": [ "uses a proxy attribute instead of the queried attribute", "compares before collecting both values" ], "guardrails": [ "never compare before both target values are collected", "never substitute a proxy attribute for the queried attribute unless equivalence is explicitly stated" ] }
```

Current skill fields: {current_skill_json}

Cluster data: {cluster_text}

Prompt M: Governance Comparison

You are performing a SkillBank governance comparison.
 You have a newly generated skill candidate and an existing skill neighbor. Your task is to determine: - Are they essentially the same and should be updated or merged? - Is the candidate a separate skill that should be added (**ADD**)? - Is the candidate significantly better and should replace the old skill (**DEPRECATE**)? - Or does the candidate lack sufficient novelty or evidence and should not be added to the library (**NOOP**)?
 Please consider the following: 1. Do their principles highly overlap? 2. Are the `when_to_use` fields within the same application boundary? 3. Are the `action_skeleton` operations using the same strategy? 4. Are `guardrails` and `known_failure_modes` complementary or conflicting? 5. Is the candidate clearer, safer, and more complete?
 Return **only a single JSON object**, in the following format:
`{ "action": "ADD", "target_skill_id": null, "deprecated_skill_ids": [], "rationale": "...", "score": 0.78 }`
 Where: - **action** can only be `ADD` / `UPDATE` / `DEPRECATE` / `NOOP` - If the action is `UPDATE`, then **target_skill_id** must be the ID of an existing skill - If the action is `DEPRECATE`, then **deprecated_skill_ids** should contain the ID of an old skill that should be deprecated - **score** is a confidence level between 0 and 1
 candidate: {candidate_json}
 neighbor: {neighbor_json}

Prompt N: Governance Conflict Detection

You are performing conflict detection in SkillBank.
 Determine if the following two skills have a **substantial conflict**.
 A substantial conflict is defined as: - opposite principles - explicitly contradictory `when_to_use` boundaries - conflicting `guardrails` - one encouraged practice is explicitly prohibited by the other
 Return only a single JSON object:
`{ "conflict": true }`
 or
`{ "conflict": false }`
 skill_a: {skill_a_json}
 skill_b: {skill_b_json}

Prompt O: Regression Evaluation

You are performing a lightweight regression test on SkillBank.
 Given a benchmark query and the skills retrieved before and after an update, determine: 1. **retrieval_correctness**: Is the retrieved skill set more reasonable and better suited to the query after the update? 2. **application_fitness**: Are the primary skills more suitable for the query after the update, are the boundaries clearer, and are the `guardrails` safer?
 Please compare **before** and **after**. No inference or benchmark inference is required. Make judgments based solely on the query and the retrieved skills.
 Return only a single JSON object, in the following format:
`{ "retrieval_better": "before", "application_better": "after", "reason": "..." }`
 Here, **retrieval_better** and **application_better** can only take the values `before` / `after` / `tie`.
 query: {query}
 before: {before_json}
 after: {after_json}

Prompt P: SkillMatch Evaluation

You are evaluating **SkillMatch** for one query-skill pair.

Given a benchmark query and one retrieved skill, determine whether this skill is **useful and appropriate** for solving the query.

A skill should be labeled as **1** if it is meaningfully useful for the query, including cases where it: - matches the task structure or task stage, - provides relevant operational guidance, - contributes useful constraints, verification rules, or procedural support.

A skill should be labeled as **0** if it is: - irrelevant to the query, - too generic to be meaningfully useful, - mismatched in task type, stage, or operational content, - unlikely to help solve the query.

Return only a single JSON object in the following format:

```
{ "label": 1, "reason": "Brief explanation." }
```

Requirements: - **label** must be either 0 or 1 - **reason** should be brief

query: {query}

skill: {skill.json}

Prompt Q: ConflictRate Evaluation

You are evaluating **ConflictRate** for one pair of retrieved skills.

Given two retrieved skills for the same benchmark query, determine whether these two skills are **mutually conflicting**.

A pair should be labeled as **1** if the two skills contain substantial conflict, including cases such as: - opposite or incompatible principles, - contradictory when-to-use boundaries, - conflicting operational guidance, - one skill encouraging behavior that the other skill explicitly forbids, - incompatible guardrails or safety constraints.

A pair should be labeled as **0** if the two skills are compatible, even if they are different in scope or emphasis.

Judge only whether the pair is conflicting. Do not judge usefulness, redundancy, or overall quality.

Return only a single JSON object in the following format:

```
{ "label": 0, "reason": "Brief explanation." }
```

Requirements: - **label** must be either 0 or 1 - **reason** should be brief

query: {query}

skill.a: {skill.a.json}

skill.b: {skill.b.json}

Prompt R: OverlapRate Evaluation

You are evaluating **OverlapRate** for one pair of retrieved skills.

Given two retrieved skills for the same benchmark query, determine whether these two skills are **substantially overlapping or redundant**.

A pair should be labeled as **1** if the two skills provide largely overlapping content, including cases such as: - highly similar applicability boundaries, - essentially the same operational procedure, - one skill being only a minor rewording or slightly narrower version of the other, - both skills contributing nearly the same guidance for this query.

A pair should be labeled as **0** if the two skills are distinct in role, boundary, or operational content, even if they are compatible.

Judge only whether the pair is substantially overlapping or redundant. Do not judge conflict, usefulness, or overall quality.

Return only a single JSON object in the following format:

```
{ "label": 1, "reason": "Brief explanation." }
```

Requirements: - **label** must be either 0 or 1 - **reason** should be brief

query: {query}

skill.a: {skill.a.json}

skill.b: {skill.b.json}

D Case study

User

You are an expert agent operating in the ALFRED Embodied Environment. Your task is to: put some spraybottle on toilet.

Retrieved Relevant Skills

- **Search Drawers Before Visible Surfaces:** When the location of a household object is unknown, first inspect nearby drawers or storage areas, then broaden the search to visible surfaces such as countertops and shelves. **When to use:** Use in pick-and-place tasks before the target object has been located, especially when it may be stored in common household containers or left on an open surface. **Action skeleton:** identify likely storage locations → inspect nearby drawers → move to countertops or shelves after empty or unproductive checks → stop searching once the target is observed. **Guardrails:** do not remain in repeatedly empty storage areas, and do not navigate to the final destination before locating the target object
- **Secure the Target Immediately Once Found:** Once the required object becomes visible, pick it up before performing additional exploration or navigating toward the destination. **When to use:** Use immediately after the target object is observed and the agent is not already holding it. **Action skeleton:** confirm that the visible object matches the task target → identify its supporting receptacle or surface → pick up the object → verify that it is in the inventory. **Guardrails:** do not leave a confirmed target behind, continue searching after finding it, or approach the destination without first securing the object
- **Deliver Directly When No State Change Is Required:** After acquiring the target in a simple placement task, navigate directly to the requested destination and place the object there if the instruction does not require cleaning, heating, cooling, or another state change. **When to use:** Use after the target object has been acquired and the task only specifies a destination without any required intermediate transformation. **Action skeleton:** inspect the instruction for state-change requirements → confirm that none are present → navigate to the destination → place the held object on or in the specified receptacle. **Guardrails:** do not introduce unnecessary cleaning or appliance interactions, and do not place the object anywhere other than the requested destination

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.

Assistant

`<think>` The task is to put a spray bottle on the toilet. To do this, I need to find a spray bottle first. The most likely places to find a spray bottle would be in drawers or on shelves. I will start by checking the drawers. `</think>`
`<action>` go to drawer 1 `</action>`

Tool Call/Response

You arrive at drawer 1. The drawer 1 is closed.

Assistant

<think> The task is to put a spray bottle on the toilet. I have just arrived at drawer 1, which is closed. Since the task involves placing an item (spray bottle) on the toilet, it makes sense to open the drawer to check if there is a spray bottle inside. If no spray bottle is found in drawer 1, I will need to explore other locations like the shelves, countertops, or other drawers. </think>
<action> open drawer 1 </action>

Tool Call/Response

You open the drawer 1. The drawer 1 is open. In it, you see nothing.

Assistant

<think> The task is to put a spray bottle on the toilet. I have already opened drawer 1 but found nothing there. The next logical step would be to examine the contents of the drawer more closely to ensure there are no hidden items. If the drawer is empty, I should then check other potential locations where a spray bottle might be found, such as other drawers, shelves, or even the toilet itself (though it's less likely to find a spray bottle there). </think>
<action> examine drawer 1 </action>

Tool Call/Response

The drawer 1 is open. In it, you see nothing.

Assistant

<think> The task is to put a spray bottle on the toilet. So far, I have checked drawer 1, which was empty. The next logical steps would be to check other potential locations where a spray bottle might be found. Based on typical household organization, spray bottles are often stored in cabinets, on shelves, or near the bathroom sink. Therefore, I will start by checking the other drawers and shelves. </think>
<action> go to drawer 2 </action>

Tool Call/Response

You arrive at drawer 2. The drawer 2 is closed.

Assistant

<think> The task is to put a spray bottle on the toilet. I have already checked drawer 1 and 2, and they were empty. Next, I should consider other places where a spray bottle might be located, such as the countertop, shelves, or other drawers. Since the task does not specify that the spray bottle needs to be cleaned or heated, I will proceed to check these locations systematically. </think>
<action> go to countertop 1 </action>

Tool Call/Response

You arrive at countertop 1. On the countertop 1, you see a mirror 1, a soapbar 1, and a spraybottle 1.

Assistant

<think> The task is to put a spraybottle on the toilet. From the observation, I am currently at the countertop 1 where I see a spraybottle 1. The next logical step is to take the spraybottle and go to the toilet to place it there. Since the spraybottle is already identified, taking it from the countertop is the next necessary action. </think>
<action> take spraybottle 1 from countertop 1 </action>

Tool Call/Response

You pick up the spraybottle 1 from the countertop 1.

Assistant

<think> I have the spraybottle in my inventory and need to place it on the toilet. According to the task, I need to put the spraybottle on the toilet. I should now navigate to the toilet and place the spraybottle there. There is no indication that the spraybottle needs to be cleaned or that the toilet requires any specific state change before placing the spraybottle. Therefore, I will proceed directly to the toilet and place the spraybottle on it. </think>
<action> go to toilet 1 </action>

Tool Call/Response

You arrive at toilet 1. On the toilet 1, you see a candle 1, a toiletpaper 2, and a toiletpaper 1.

Assistant

<think> I am now at the toilet and I have the spraybottle in my inventory. The task is to put the spraybottle on the toilet. Since there are no guardrails that require me to clean or change the state of the spraybottle, I can proceed directly to placing it on the toilet. There is no need to examine either the spraybottle or the toilet further as the action is straightforward. </think>
<action> move spraybottle 1 to toilet 1 </action>

Tool Call/Response

You move the spraybottle 1 to the toilet 1.