SkillX: Automatically Constructing Skill Knowledge Bases for Agents

Anonymous ACL submission

Abstract

Learning from experience is critical for building capable large language model (LLM) agents, yet prevailing self-evolving paradigms remain inefficient: agents learn in isolation, repeatedly rediscover similar behaviors from limited experience, resulting in redundant exploration and poor generalization. To address this problem, we propose **`SkillX`**, a fully automated framework for constructing a **plug-and-play skill knowledge base** that can be reused across agents and environments. `SkillX` operates through a fully automated pipeline built on three synergistic innovations: *(i) Multi-Level Skills Design*, which distills raw trajectories into three-tiered hierarchy of strategic plans, functional skills, and atomic skills; *(ii) Iterative Skills Refinement*, which automatically revises skills based on execution feedback to continuously improve library quality; and *(iii) Exploratory Skills Expansion*, which proactively generates and validates novel skills to expand coverage beyond seed training data. Using a strong backbone agent (GLM-4.6), we automatically build a reusable skill library and evaluate its transferability on challenging long-horizon, user-interactive benchmarks, including AppWorld, BFCL-v3, and τ^2-Bench. Experiments show that `SkillX` consistently improves task success and execution efficiency when plugged into weaker base agents, highlighting the importance of structured, hierarchical experience representations for generalizable agent experience learning.

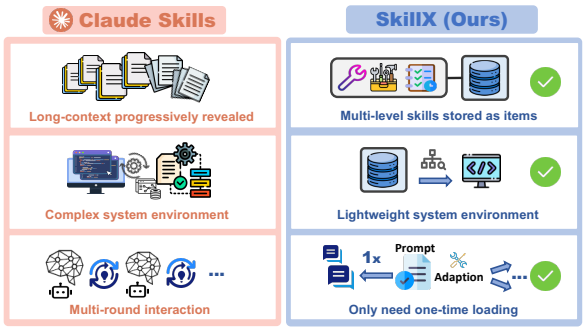


Figure 1: Claude Skills follow a long-context, progressively disclosed format, which requires a complex sandboxing system and multiple interactions, thereby posing challenges to robust reasoning. In contrast, SkillX adopts a hierarchical, itemized representation that can be stored and retrieved via a lightweight retrieval module and injected into the system prompt, making it easier to transfer across base models.

1 Introduction

Large language model (LLM) based agents (OpenAI, 2025; DeepSeek-AI, 2025; Team et al., 2025b; Yang et al., 2025) have recently demonstrated remarkable progress in long-horizon decision making with tools, enabling complex behaviors such as API calling (Trivedi et al., 2024; Patil et al., 2025; Li et al., 2025), web navigation (Yao et al., 2023; Zhou et al., 2024; Mialon et al., 2023), scientific discovery (Ou et al., 2025; Liu et al., 2025; Qiao et al., 2025; Novikov et al., 2025), and interactive assistants (Barres et al., 2025; Yao et al., 2024; He et al., 2025). Despite these advances, most agents still approach each new task largely *from scratch*, relying on direct reasoning or limited task-specific demonstrations. This paradigm is costly, brittle, and fundamentally at odds with how intelligent systems are expected to accumulate and reuse experience over time.

A natural resolution is to enable agents to ***learn from experience*** (Sutton, 2025). Recent work has explored self-evolving agents that iteratively reflect on past executions and improve their behavior over time (Wang et al., 2025c; Fang et al., 2025c; Zhao et al., 2024; Xu et al., 2025; Cao et al., 2025). While promising, these approaches often fail to deliver scalable and transferable gains. In practice, experience learning typically suffers from three structural limitations. *(1) Isolated Learning*: agents execute the same tasks repeatedly and re-extract similar experiences independently, leading to substantial redundancy. *(2) Weak Generalization of Experience*: in complex environments, high-

quality training data are scarce, so the mined experiences often transfer poorly to new tasks. *(3) Model Capability Bottleneck*: when experience is harvested solely through an agent's own exploration and reflection, what can be extracted is ultimately capped by the agent's current capability frontier. These challenges point to a more fundamental question: **What form of experience can be broadly reusable across agents of varying capabilities and across diverse environments?**

Existing work has proposed multiple representations of experience, such as insights (Cao et al., 2025; Ouyang et al., 2025), workflows (Wang et al., 2025c,b; Han et al., 2025), or trajectories (Zhao et al., 2024; Fang et al., 2025c). However, none of these representations simultaneously offer strong transferability, efficient retrieval, and direct executability. Inspired by Claude Skills (Anthropic, 2025), we argue that **skills** provide a more suitable abstraction: they encapsulate reusable competencies that directly support task execution. Nonetheless, prior skill-based designs often rely on long-context, progressive disclosure, which place heavy demands on reasoning and environment instrumentation, limiting robustness and practical reuse, as illustrated in Figure 1.

In this work, we introduce `SkillX`, a fully automated framework for constructing a **plug-and-play skill knowledge base** from agent experience. Our core insight is that transferable experience should be organized **hierarchically**, rather than as monolithic behaviors. `SkillX` therefore represents experience at three complementary levels: *(i) Planning Skills*, which capture high-level task organization; *(ii) Functional Skills*, which implement reusable, tool-based subroutines; and *(iii) Atomic Skills*, which encode execution-oriented usage patterns and constraints. This multi-level design yields skills that are concise, composable, and robust to distributional shifts. `SkillX` builds such a skill library through a fully automated pipeline. A strong backbone agent first performs rollouts on training tasks and distills multi-level skills from successful trajectories. The extracted skills are then **iteratively refined** through consolidation and validation, improving library quality over time. Finally, `SkillX` performs **experience-guided exploration** to expand the skill space by targeting under-utilized tools and failure-prone behaviors, enabling generalization beyond the initial training distribution.

To build a reliable, plug-and-play skill library, we instantiate `SkillX` with a strong agent, GLM-4.6 (Team et al., 2025a), and pre-build a skill library on challenging, user-interactive, long-horizon benchmarks, including: AppWorld (Trivedi et al., 2024), BFCL-v3 (Patil et al., 2025), and τ^2-Bench (Barres et al., 2025). Our experiments show that this plug-and-play skill library can be directly plugged into base agents (e.g., Qwen3-32B (Yang et al., 2025)), yielding around a 10% performance improvement while also improving execution efficiency. We further demonstrate the advantages of our multi-level skill design for experience representation, and show that both iterative refinement and skill expansion provide additional gains. In a nutshell, we conclude our contributions as:

- We propose a hierarchical skill representation that transforms raw trajectories into reusable planning, functional, and atomic skills.
- We present `SkillX`, a fully automated and extensible framework for pre-building plug-and-play skill libraries for LLM agents, featuring iterative refinement and skill expansion.
- We release the resulting plug-and-play skill library and provide empirical evidence across multiple agent benchmarks that it can directly enhance the capabilities of weaker agents.

2 Preliminaries

Agent Definition We consider a general interactive setting where an agent solves tasks by acting in an environment. An environment is defined as $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{P})$, where $\mathcal{A}$ is the set of executable actions, $\mathcal{S}$ is the set of observable states, and $\mathcal{P}(s' \mid s, a)$ is the transition dynamics. At time step t, the agent receives an observation $o_t \in \mathcal{O}$ and produces an action $a_t \in \mathcal{A}$. Following the ReAct style formulation, the agent therefore selects an action $\hat{a}_t \in \hat{\mathcal{A}}$ conditioned on its context $c_t = (o_1, \hat{a}_1, \ldots, o_{t-1}, \hat{a}_{t-1}, o_t)$: $\hat{a}_t \sim \pi(\cdot \mid c_t)$. Executing $\hat{a}_t \in \mathcal{A}$ yields a new observation via the environment. The final trajectory is $\tau = (o_1, \hat{a}_1, \ldots, o_T, \hat{a}_T)$.

LLM Agent and Skill-Conditioned Execution. Let $\mathcal{Q}$ be the tasks set. We write $q \in \mathcal{Q}$ for sampling a task, and let $R(\tau, q) \in \{0, 1\}$ be a task-dependent success indicator. We model the LLM agent as a policy π that induces a trajectory distribution. Without external skills, the agent generates trajectories by direct reasoning: $\tau \sim \pi(\cdot \mid q)$.

To reduce redundant exploration and improve task completion, we equip the agent with a *skills library* $\mathcal{D} = \{s_1, \ldots, s_{|\mathcal{D}|}\}$ and a *skill retriever* that recalls a set of relevant skills for the current task. Concretely, given $q \in \mathcal{Q}$, a retrieval function (typically implemented via semantic-similarity retrieval) ρ, returns a skill subset $\mathcal{S}_q = \rho(q), \mathcal{S}_q \subseteq \mathcal{D}$. The LLM agent then generates a trajectory by conditioning on the retrieved skill set: $\tau' \sim \pi(\cdot \mid \mathcal{S}_q, q)$. Our objective is to design the skills library $\mathcal{D}$ and the usage within π such that the expected success rate is improved: $\mathbb{E}_{q\in\mathcal{Q},\, \tau'\sim\pi(\cdot|\mathcal{S}_q,q)} R(\tau', q) > \mathbb{E}_{q\in\mathcal{Q},\, \tau\sim\pi(\cdot|q)} R(\tau, q)$.

3 SkillX Design and Implementation

3.1 Multi-Level Skills Design

In tool-centric agent scenarios, we structure the skills required by the model into three levels (see Figure 2): $\mathcal{D} = S_{\text{plan}} \oplus S_{\text{func}} \oplus S_{\text{atomic}}$. corresponding to planning skills, functional skills, and atomic skills, respectively. In a given environment $\mathcal{E}$, let $\mathcal{T}$ denote the set of tool actions. *(i) Atomic skill* s_{atomic} is aligned with a single tool $t \in \mathcal{T}$ and is modeled as an extended semantic specification of t, e.g., as enriched descriptions, constraints, or usage patterns that refine the effective behavior of t. *(ii) Functional skill* s_{func} abstracts a subtask and can be regarded as a macro-operation that accomplishes a sub-query. We assume each task q admits a decomposition into n subtasks, $\{q_{\text{subtask},1}, q_{\text{subtask},2}, \ldots, q_{\text{subtask},n}\}$ and each s_{func} corresponds to skills to accomplish $q_{\text{subtask},i}$. Specifically, s_{func} is grounded in a set of tool actions, which can be instantiated as a composition of tools $\mathcal{T}_{\text{func}} \subseteq \mathcal{T}$. *(iii) planning skill* s_{plan} aligns with the organizational structure of the subtasks (e.g., ordering, dependencies, and branching), specifying how functional skills should be composed to solve q. Next, we describe the extraction methods for the three skill levels.

3.2 Rollout and Skills Extraction

Given a task q, we first perform m-sized rollouts, reusing the agent's inference procedure to collect trajectories. We then extract the multi-level skills from these trajectories, with skill extractor f.

Planning Skills Extraction. Given a successful trajectory, we extract the planning skill s_{plan} by compressing the trajectory into an ordered set of high-level steps. During this compression, we explicitly filter out non-essential transitions such as exploration, backtracking, and trial-and-error behaviors that are incidental to the final solution but detrimental to skill reuse. Moreover, for excessively long or verbose environment feedback, we apply summarization to obtain compact state descriptions, which improves the stability and fidelity of the extracted high-level skills.

Functional Skills Extraction. We leverage the previously extracted planning skill s_{plan} to guide the extraction of functional skills. Concretely, given a plan and its corresponding trajectory, we iteratively prompt the model to extract the functional skill s_{func} that aligns with the objective of each subtask $q_{\text{subtask},i}$. Formally, each s_{func} is represented with three key fields: `name` (the skill name), `document` (a description of inputs, outputs and usage notes), and `content` (the tool invocation pattern for completing subtask $q_{\text{subtask},i}$).

Atomic Skills Extraction. Atomic skills are single tool specifications that extend the original tool schema with reusable, execution-oriented usage patterns. They serve as a low-level complement when higher-level functional skills s_{func} are missing or incomplete. We prompt the model to distill s_{atomic} from trajectories the invocation patterns, typical parameter configurations, and practical notes, especially constraints and common failure modes observed in real usage. The representation of s_{atomic} is unified with s_{func}.

3.3 Iterative Skills Refinement

With only a limited amount of seed training data, a key question is whether we can maximize the utility of the available supervision to extract additional skills and continuously improve existing ones. Inspired by prior works (Cai et al., 2025b,a; Yuksekgonul et al., 2024), we adopt a text-based iterative optimization paradigm for the skill library. Concretely, at k-th iteration, we start from the current skill library $\mathcal{D}^{(k)}$, repeatedly rollouts from the training set, then extract multi-level skills. We subsequently apply a refinement operator ϕ, including: *Skills Merge* and *Skills Filter*. Finally, we update the skill library $\mathcal{D}^{(k)}$ with the refined skills to obtain skill library $\mathcal{D}^{(k+1)}$, including three update operations: `add`, `modify` or `keep`.

Iterative Skills Library Construction. We construct the skill library in an iterative manner. Let

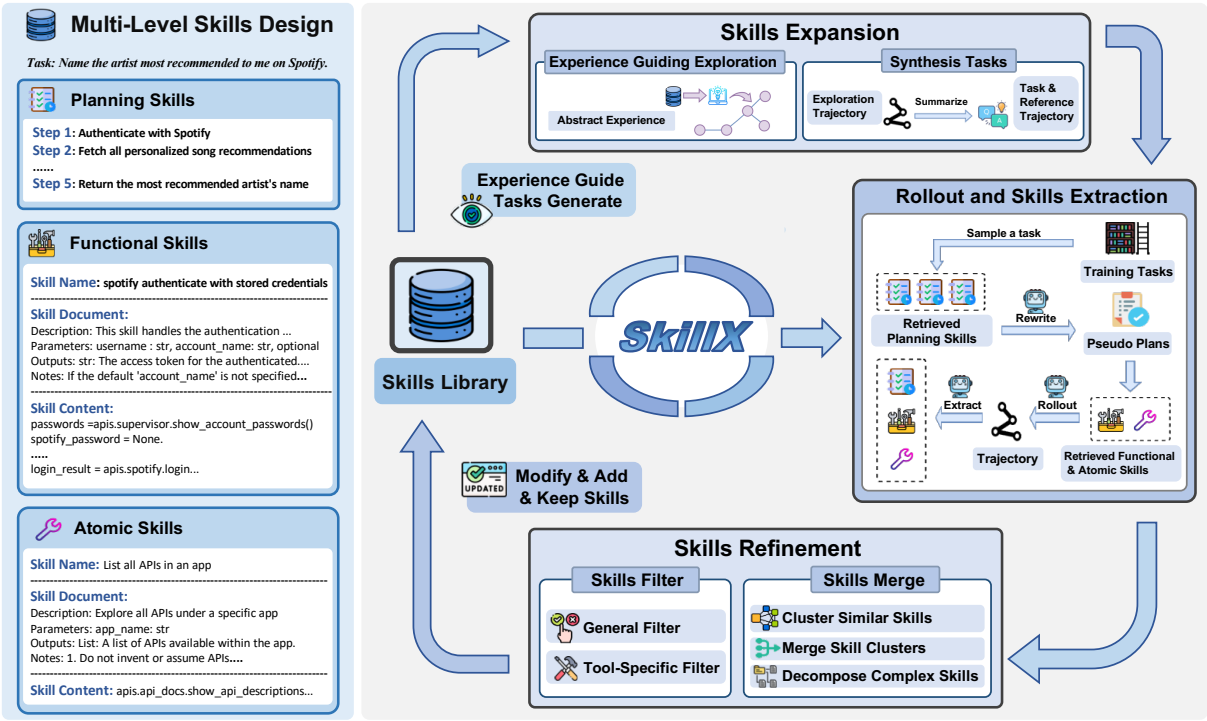


Figure 2: `SkillX` provides an automated, iterative pipeline for constructing a skills library, integrating skills extraction. skills expansion and skills refinement. The skills library is organized into three levels: planning skills, functional skills, and atomic skills.

$\mathcal{D}^{(0)} = \emptyset$ be an initial empty library. In iteration $k = 0, 1, \ldots,$ we roll out the agent augmented with the current library $\mathcal{D}^{(k)}$ on tasks sampled from the training set $\mathcal{Q}_{\text{train}}$ to obtain a set of trajectories

$$\tau^{(k)} \sim \pi(\cdot \mid \rho_{\mathcal{D}^{(k)}}(q), q), \quad q \in \mathcal{Q}_{\text{train}}, \quad (1)$$

and denote $\mathcal{K}^{(k)} = \{\tau_1^{(k)}, \ldots, \tau_{N_k}^{(k)}\}$. A skill extractor f produces a variable-size set of candidate skills from each trajectory, $\mathcal{S}_i^{(k)} = f(\tau_i^{(k)})$ and we aggregate all the skills extracted from the batch via $\mathcal{S}^{(k)} = \bigcup_{i=1}^{N_k} \mathcal{S}_i^{(k)}$. Additionally, we define a refinement operator ϕ to merge and filter the skills. The library is then updated as

$$\mathcal{D}^{(k+1)} \triangleq \mathcal{D}^{(k)} \cup \phi\left(\mathcal{S}^{(k)}\right) = \mathcal{D}^{(k)} \cup \phi\left(\bigcup_{i=1}^{N_k} \mathcal{S}_i^{(k)}\right). \quad (2)$$

Let $\mathcal{Q}_{\text{test}}$ denote a test distribution. We aim to iteratively improve the library such that the performance of the induced skill-conditioned agent is maximized on $\mathcal{Q}_{\text{test}}$:

$$\max_k \ \mathbb{E}_{q \sim \mathcal{Q}_{\text{test}}}\left[\mathbb{E}_{\tau \sim \pi(\cdot \mid \rho_{\mathcal{D}^{(k)}}(q), q)}\left[R(\tau, q)\right]\right], \quad (3)$$

and we stop the iteration when this test performance no longer improves.

Skills Merge. After extracting skills from each trajectory, we often obtain many functionally redundant skills that, despite surface differences, correspond to the same underlying skill pattern. How to update a single skill when multiple heterogeneous update directions are available? We merge skills from an optimization-based perspective. For a specific skill s with current embedding, we first retrieve and cluster a set of semantically similar skills using cosine similarity. The resulting cluster can be interpreted as providing multiple complementary update directions for the same underlying skill, a multi-dimensional refinement of s. Let $\mathcal{Z}(s) = \{1, \ldots, z\}$ index the semantically similar skills associated with skill s. Each neighbor i induces a candidate update direction δ_i, yielding a candidate updated state $s_i' = s + \delta_i, \ i \in \mathcal{Z}(s)$.

We then aggregate these candidate directions into the final direction. The simplest form is to merge the directions: $\delta_{\text{agg}} = \sum_{i \in \mathcal{Z}(s)} \delta_i$. The final update is applied as $s^+ = s + \delta_{\text{agg}}$. Specifically, we treat semantically similar skills as multiple update views of the same skill, and use the combined direction as the final update direction. Finally, we merge semantically similar skills into a single one. If the merged skill gets overly complex, we decompose it to more modular, reusable skills.

Skills Filter. We enforce skill quality via a strict two-stage filtering procedure. *(1) General Filter.* This stage removes skills unlikely to be portable or compositional, including those that depend on extraneous Python packages, expose idiosyncratic function-style definitions, or overly-encapsulated skills. *(2) Tool-specific Filter.* This stage mitigates tool-use hallucinations by validating each skill against the environment-provided tool schema, rejecting skills that reference non-existent tools, invalid parameters, or schema-incompatible arguments. Together, these filters maintain a high-precision skill library while preserving flexibility across heterogeneous agent benchmarks.

Skills Library Update. After completing *Skill Merge* and *Skill Filter*, we perform concrete updates to the skill library $\mathcal{D}^k$ for the k-th iteration, including three types: `add` new skills, `modify` existing skills, and `keep` skills unchanged. Furthermore, the entire pipeline can be executed iteratively over multiple rounds. Through this continual update process, the skill library progressively improves in coverage, quality, and compositional richness, enabling increasingly effective skill reuse for downstream agent tasks.

3.4 Exploratory Skills Expansion

While skills distilled from a seed training set $\mathcal{Q}_{\text{train}}$ can already improve an agent's performance, relying solely on scarce demonstrations is insufficient in complex environments with large tool spaces (e.g., (Trivedi et al., 2024) exposes hundreds of APIs). Inspired by Zhai et al. (2025), we adopt an *Experience Guiding Exploration* scheme to broaden coverage beyond what is observed in the seed data, encouraging the agent to interact with the environment and exercise a wider range of tools. We guide exploration using experience collected from rollouts on the seed set (e.g., tools the agent already uses reliably, tools with high failure rates, and tools that are never invoked), thereby prioritizing under-explored or failure-prone tools to improve sample efficiency. After collecting exploratory trajectories, we synthesize new tasks $\mathcal{Q}_{\text{syn}}$ from these interactions, and then rerun our skill acquisition and refinement pipeline on the resulting data to iteratively expand the skill library. Compared to the previous random exploration strategy (Zhai et al., 2025), our approach discovers a more diverse set of skills.

4 `SkillX` Usage

Planning Skills Retrieval and Pseudo-Plan Rewriting. For a novel and complex agent task q, directly retrieving past experiences based solely on task similarity may lead to a mismatch between retrieved experiences and the actual execution trajectory. This issue becomes particularly pronounced in environments where execution dynamics are strongly influenced by user profiles, contextual constraints, or other external factors. To improve retrieval relevance, inspired by (Gao et al., 2022), we first retrieve high-level planning skills associated with similar tasks $\mathcal{P}(q) = \rho(q)$, where ρ is a similarity retrieval function and $\mathcal{P}(q)$ is the retrieved planning skills. Then we prompt the model to self-rewrite a task-specific pseudo-plan conditioned on the current task $\tilde{p}(q) = \text{LLM}_{\text{rewrite}}\big(q,\, \mathcal{P}(q)\big)$. This rewritten pseudo-plan serves as an intermediate retrieval query to better align subsequent skill retrieval with the current execution setting. To mitigate hallucination risks and prevent speculative content from affecting agent behavior, the pseudo-plan is not injected into the final system prompt.

Functional and Atomic Skills Retrieve. Given the rewritten pseudo-plan $\tilde{p}(q) = \{\text{step}_1, \text{step}_2, \ldots, \text{step}_p\}$, we treat each step as a retrieval query to retrieve functional and atomic skills. For step_i, we first retrieve relevant skills $\mathcal{S}_i = \rho(\text{step}_i)$ and then remove duplicates across steps, $\mathcal{S}' = \text{dedup}\Big(\bigcup_{i=1}^{p} \mathcal{S}_i\Big)$. To keep the context concise and task-relevant, we further ask the LLM to self-filter the retrieved candidates and retain only applicable skills $\mathcal{S}_q = \text{LLM_select}(q, \tilde{p}(q), \mathcal{S}')$, where $\mathcal{S}_q$ is the final skill set used for solving the query q.

5 Experiment

5.1 Experimental Settings

Benchmarks and Metrics. We conduct the evaluation on complex, long-horizon, user-interactive agent benchmarks, including BFCL-v3 (Patil et al., 2025), AppWorld (Trivedi et al., 2024), and τ^2-bench (Barres et al., 2025). For BFCL-v3, we use the base multi-turn category and randomly split it into 50 training instances and 150 test instances. AppWorld provides 90 training instances and the Test Normal category as test set. τ^2-bench defines training and test splits for each sub-domain. Additional details are provided in the Appendix A.1. For AppWorld and BFCL-v3, we report Avg@4

Model	Methods	BFCL-V3		AppWorld		τ^2-Bench		
		Avg@4	Pass@4	Avg@4	Pass@4	Retail	Airline	Telecom
Qwen3-32B	No Memory*	53.67	73.33	27.68	47.62	53.75	38.75	36.25
	A-Mem*	53.67	73.00	26.79	50.59	53.12	38.75	38.12
	AWM*	55.67	76.00	30.80	55.95	55.00	40.00	38.12
	AWM‡	56.67	76.33	34.45	56.25	57.50	41.25	40.62
	ExpeL*	57.33	77.67	32.87	58.93	56.25	42.50	39.38
	ExpeL‡	59.33	78.83	32.94	58.78	58.12	43.75	41.25
	SkillX‡	**63.67**	**82.00**	**35.12**	**58.93**	**66.87**	**47.50**	**43.75**
Kimi-K2-Instruct-0905	No Memory*	65.17	78.00	46.88	70.24	75.62	51.25	78.12
	A-Mem*	65.17	76.67	46.58	72.62	76.25	52.50	76.87
	AWM*	65.33	79.00	49.70	76.19	76.25	53.75	77.50
	AWM‡	64.67	79.17	50.60	76.49	76.25	53.75	77.50
	ExpeL*	66.33	79.33	52.53	78.57	77.50	55.50	78.75
	ExpeL‡	66.00	79.67	52.98	78.87	77.50	56.25	79.37
	SkillX‡	**66.83**	**81.33**	**56.40**	**81.55**	**78.12**	**58.75**	**82.50**
GLM-4.6	No Memory*	76.67	83.33	60.27	83.33	76.25	70.00	70.63
	A-Mem*	76.50	83.00	60.57	83.93	76.88	70.00	68.75
	AWM*	77.17	84.00	62.20	84.52	77.50	71.25	70.63
	ExpeL*	78.83	85.33	64.14	85.12	77.50	72.50	71.25
	SkillX*	**79.50**	**86.00**	**64.88**	**88.69**	**82.50**	**76.25**	**71.88**

Table 1: Main results of SkillX on three benchmarks. Methods with $*$ mean that the experience extraction model is aligned with the inference model. Methods with ‡ mean that GLM-4.6 is used for experience extraction, while inference still relies on the original model.

and Pass@4, the average success rate over four independent runs and the probability of succeeding at least once across four runs, respectively. Following the (Barres et al., 2025) evaluation setup, we report Pass^1, the pass rate over running four times.

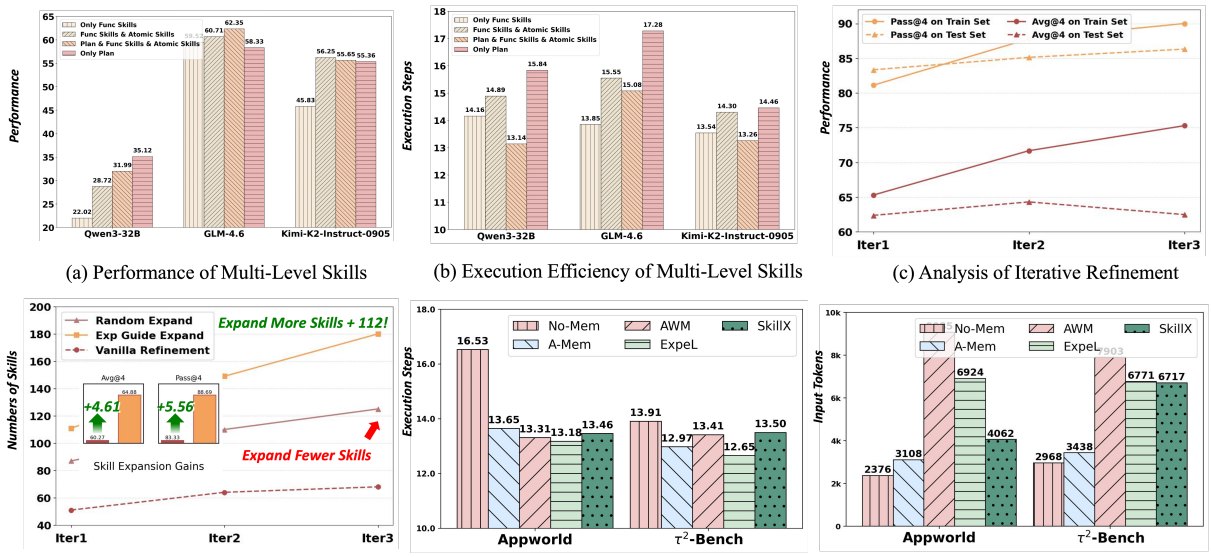
Models and Baselines. To assess the effectiveness of SkillX, we evaluate three Agentic base models that vary in model size and reasoning style (thinking and non-thinking), including Qwen3-32B (Yang et al., 2025), Kimi-K2-Instruct-0905 (Team et al., 2025b), and GLM-4.6 (Team et al., 2025a). We compare against four representative baselines: (1) No-memory; (2) A-Mem (Xu et al., 2025); (3) AWM (Wang et al., 2025c); and (4) ExpeL (Zhao et al., 2024). For a fair comparison, all methods retrieve experience only based on the user's initial query and insert the retrieved content into the system prompt following a unified protocol. Full baseline details are provided in the Appendix A.2.

5.2 Main Results

SkillX Boost Agentic Performance of Base LLMs. As shown in Table 1, SkillX improves the base model's performance. In particular, Qwen3-32B gains roughly around 10 points across multiple benchmarks. For K2 (Kimi-K2-Instruct-0905), we observe a clear improvement on AppWorld, whereas the gains are modest on the other two tool call intensive benchmarks. We infer this is because K2 relies more heavily on the original tool schema and does not effectively leverage the additional contextual information.

Multi-Level Skills Design Outperform Other Forms of Experience Representation. When the experience extraction model is aligned with the execution model, SkillX consistently outperforms all baseline methods, as indicated by the methods with $*$ in Table 1. Among them, ExpeL retrieves past trajectories and uses them as few-shot demonstrations, which provides a more direct performance gain than the other baselines. However, the agent capability required for multi-level skill decoupling offers a more advantageous form of experience representation.

Suboptimal Experience Representations Hinder Transfer Performance. We further evaluate the GLM-4.6 extracted experience with AWM and ExpeL on the weaker models, see the results of methods with ‡ in Table 1. However, the performance still lagged behind that of SkillX. This indicates that **distilling experience from a strong model is effective, but the form of experience representation is even more critical**. Consequently,



(a) Performance of Multi-Level Skills (b) Execution Efficiency of Multi-Level Skills (c) Analysis of Iterative Refinement

(d) Skill Expansion Strategies Analysis (e) Average Execution Steps (f) Average Input Tokens

Figure 3: **Comprehensive Analysis of `SkillX`. (a) Performance of Multi-skills:** Models exhibit varying performance under different skill composition. **(b) Execution Efficiency of Multi-skills:** Jointly composing all skills yields the best execution efficiency. **(c) Iterative Optimization:** Iterative skill refinement further improves performance. **(d) Skill Expansion Strategies:** Experience-guided expansion achieves the best on scalability and performance gains. **(e) Analysis of Execution Steps:** Experience-based learning reduces the number of execution steps. **(f) Analysis of Input Tokens:** Properly balancing input tokens is crucial for controlling inference cost.

suboptimal experience representation can hinder effective experience transfer. These results further demonstrate the advantage of `SkillX` in transferring experience across base models.

5.3 Analysis

Which skill is more effective? We analyze the effects of multi-level skills on AppWorld (Figure 3(a,b)). *(i) Planning skills* consistently reduce execution steps, with larger gains for weaker models (e.g., Qwen3-32B and K2), especially when combined with Functional Skills. For Qwen3-32B, however, adding Functional and Atomic Skills can hurt performance, likely because the model over-relies on retrieved skills rather than adapting them. For stronger models, pseudo-planning may be less effective in complex environments. *(ii) Functional skills* contribute the largest performance gains, indicating that they are an effective form of experience abstraction. *(iii) Atomic skills* are crucial for clarifying key APIs; removing them causes substantial drops, highlighting their role in complementing tool schemas and covering missing APIs. Overall, GLM-4.6 benefits most from the full skill set, K2 from Functional + Atomic Skills, and Qwen3-32B from Planning Skills alone, showing that multi-level skills provide complementary support for diverse agent models.

Iterative Refinement Strategies Further Enhances `SkillX` Performance. We evaluate effectiveness of multi-round iterative refinement for the skill library of `SkillX` on AppWorld (Figure 3 (c)). Overall, multiple iterations further improve performance on both training and test sets. Leveraging existing training data, the process continually improves various aspects of skills, such as documentation and content. Besides, it can slightly expand the size of the skill library (Figure 3 (d)). However, when training data are limited, text-only optimization can lead to overfitting. Thus, selecting an appropriate number of update rounds is crucial to obtain a higher-quality skill library.

Skill Expansion Strategies Improve Generalization. We compare two skill expansion strategies: *random exploration* and *experience-guided* expansion, as shown in Figure 3 (d). In terms of skill growth, the experience-guided strategy yields substantially more novel skills, as random exploration treats past executions in isolation and repeatedly rediscovers already identified skills. Empirically, the experience guided strategy yields performance improvement through skill expansion. Overall, our results indicate that in complex environments, particularly under scarce training data, skill expansion is a crucial component of experience learning.

SkillX Enhances Agent Execution Efficiency. Learning from experience not only improves the performance of the base model, but also enhances the execution efficiency of the agent. Our experiments further corroborate this effect (see Figure 3 (e) and Figure 3 (f)). Although we do not achieve the minimum number of execution steps or the fewest input tokens, we obtain the best overall performance (see Table 1). These results further highlight the advantages of our multi-level skill design and skills library construction.
We also provide Further Analysis in Appendix B.

6 Related Work

Encoding For Agent Experience. With the advent of the experience era (Sutton, 2025), agents can achieve self-evolving (Gao et al., 2025; Fang et al., 2025a; Xia et al., 2026) by encoding past experience and reusing it in context (Dou et al., 2026) to guide future behavior. Existing approaches to text token-level experience encoding (Zhang et al., 2025b; Hu et al., 2025) can be broadly grouped into three categories: *(i) Case-based Experience*: Agents directly store successful trajectories and retrieve them later as few-shot in context (Zhao et al., 2024; Zheng et al., 2024; Zhou et al., 2025). *(ii) Strategy-based Experience*: By summarizing and contrasting successful versus failed trajectories, agents distill highe-level insights (Cao et al., 2025; Ouyang et al., 2025; Cai et al., 2025a; Wang et al., 2025c; Tang et al., 2025; Zhang et al., 2025a). *(iii) Skill-based Experience*: Trajectories are segmented and distilled into modular, reusable skills, such as textual skills or programmatic skills (Wang et al., 2025b,a, 2024; Fang et al., 2025c; Han et al., 2025; Chen et al., 2026; Zheng et al., 2026; Wang et al., 2026a; Zhou et al., 2026a; Zhang et al., 2026b; Ni et al., 2026; Zhou et al., 2026b). However, it remains unclear which unified experience representation is both easily pluggable and consistently effective, especially in diverse and complex agentic tool-use scenarios (Trivedi et al., 2024; Yao et al., 2024; Patil et al., 2025; Barres et al., 2025; He et al., 2025; Li et al., 2025; Zheng et al., 2025; Jiang et al., 2026; Xing et al., 2026; Li, 2026; Li et al., 2026). In this work, we adopt a hybrid representation, high-level planning coupled with textual skills, which yields substantial improvements for the base model.

Agent Experience Knowledge Base Construction. The construction pipeline of an Experience Knowledge Base typically consists of two steps: static construction and online adaptation. *(i) Static construction* repeatedly performs independent runs over training sets or human-curated information sources, extracts experience from the resulting trajectories, and iteratively refines the knowledge base until performance plateaus (Zhang et al., 2025c; Cai et al., 2025b; Anthropic, 2025; Wang et al., 2026b; Gallego, 2026; Yang et al., 2026a). *(ii) Online Adaptation* updates the Experience Knowledge Base in real time after each batch of newly executed tasks, enabling adaptation to continuously evolving test environments (Latimer et al., 2025; Fang et al., 2025b; Cao et al., 2025; Du et al., 2025; Yang et al., 2026b; Yao et al., 2025; Zhang et al., 2026a; Liang et al., 2026; Zhang et al., 2026a).

While dynamic updating is central to continual learning from experience, pre-building a strong static Experience Knowledge Base remains necessary in practice. However, under the task-scarcity challenge in complex agent settings (Patil et al., 2025; Barres et al., 2025; He et al., 2025; Li et al., 2025), we further extend skills by combining task synthesis (Zhai et al., 2025; Mai et al., 2025; Shi et al., 2025; Ramrakhya et al., 2025; Guo et al., 2025) to construct more challenging tasks. To our knowledge, this is the first work to provide a directly reusable skill knowledge base together with an automated pipeline for skill construction.

7 Conclusion

We introduced SkillX, an automated framework for building a plug-and-play skill library for LLM-based agents. To enable more efficient experience transfer, we design multi-level skills, including planning skills, functional skills, and atomic skills from the perspective of tool granularity. SkillX iteratively refines and expands the library through three core components: *(i) skills extraction*, which rolls out an agent with the current library and extracts multi-level skills; *(ii) skills refinement*, which iteratively improves skills using execution feedback, while maintaining quality via skill merging and strict filtering; and *(iii) exploratory skills expansion*, which proactively broadens coverage beyond the seed training set. Our experiments demonstrate that SkillX transfers effectively to other models and provides advantages in experience representation. Finally, we will release the optimized skill library constructed by SkillX to facilitate further community exploration.

Limitations

Cross-environment transfer. SkillX is currently most naturally applicable when skills can be grounded in a relatively stable tool environment. The extracted skills are associated with specific tool schemas, which makes direct reuse across substantially different domains or tool ecosystems less straightforward.

User-interactive settings. The current study focuses mainly on tool-using agent environments. More user interactive scenarios, particularly dialogue scenarios without function calls, are not yet the primary focus of this work.

References

Anthropic. 2025. Equipping agents for the real world with agent skills. Anthropic Engineering Blog.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-bench: Evaluating conversational agents in a dual-control environment. *Preprint*, arXiv:2506.07982.

Yuzheng Cai, Siqi Cai, Yuchen Shi, Zihan Xu, Lichao Chen, Yulei Qin, Xiaoyu Tan, Gang Li, Zongyi Li, Haojia Lin, Yong Mao, Ke Li, and Xing Sun. 2025a. Training-free group relative policy optimization. *CoRR*, abs/2510.08191.

Zhicheng Cai, Xinyuan Guo, Yu Pei, Jiangtao Feng, Jiangjie Chen, Ya-Qin Zhang, Wei-Ying Ma, Mingxuan Wang, and Hao Zhou. 2025b. FLEX: continuous agent evolution via forward learning from experience. *CoRR*, abs/2511.06449.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. 2025. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. *Preprint*, arXiv:2512.10696.

Tianyi Chen, Yinheng Li, Michael Solodko, Sen Wang, Nan Jiang, Tingyuan Cui, Junheng Hao, Jongwoo Ko, Sara Abdali, Leon Xu, Suzhen Zheng, Hao Fan, Pashmina Cameron, Justin Wagle, and Kazuhito Koishida. 2026. Cua-skill: Develop skills for computer using agent. *Preprint*, arXiv:2601.21123.

DeepSeek-AI. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *CoRR*, abs/2512.02556.

Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, Huaibing Xie, Jianglu Hu, Shaolei Wang, Weichao Wang, Yanling Xiao, Yiting Liu, Zenan Xu, Zhen Guo, Pluto Zhou, Tao Gui, Zuxuan Wu, Xipeng Qiu, Qi Zhang, Xuanjing Huang, Yu-Gang Jiang, Di Wang, and Shunyu Yao. 2026. Cl-bench: A benchmark for context learning. *Preprint*, arXiv:2602.03587.

Xingbo Du, Loka Li, Duzhen Zhang, and Le Song. 2025. Memr^3: Memory retrieval via reflective reasoning for llm agents. *Preprint*, arXiv:2512.20237.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, Zhaochun Ren, Nikos Aletras, Xi Wang, Han Zhou, and Zaiqiao Meng. 2025a. A comprehensive survey of self-evolving AI agents: A new paradigm bridging foundation models and lifelong agentic systems. *CoRR*, abs/2508.07407.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. 2025b. Lightmem: Lightweight and efficient memory-augmented generation. *Preprint*, arXiv:2510.18866.

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

Víctor Gallego. 2026. Distilling feedback into memory-as-a-tool. *Preprint*, arXiv:2601.05960.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. 2025. A survey of self-evolving agents: On path to artificial super intelligence. *CoRR*, abs/2507.21046.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2022. Precise zero-shot dense retrieval without relevance labels. *Preprint*, arXiv:2212.10496.

Jiacheng Guo, Ling Yang, Peter Chen, Qixin Xiao, Yinjie Wang, Xinzhe Juan, Jiahao Qiu, Ke Shen, and Mengdi Wang. 2025. Genenv: Difficulty-aligned co-evolution between llm agents and environment simulators. *Preprint*, arXiv:2512.19682.

Dongge Han, Camille Couturier, Daniel Madrigal Díaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. 2025. Legomem: Modular procedural memory for multi-agent LLM systems for workflow automation. *CoRR*, abs/2510.04851.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao. 2025. Vitabench: Benchmarking llm agents with versatile interactive tasks in real-world applications. *Preprint*, arXiv:2509.26490.

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. 2025. Memory in the age of ai agents. *Preprint*, arXiv:2512.13564.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. 2026. Xskill: Continual learning from experience and skills in multimodal agents. *Preprint*, arXiv:2603.12056.

Chris Latimer, Nicoló Boschi, Andrew Neeser, Chris Bartholomew, Gaurav Srivastava, Xuan Wang, and Naren Ramakrishnan. 2025. Hindsight is 20/20: Building agent memory that retains, recalls, and reflects. *Preprint*, arXiv:2512.12818.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. 2025. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *Preprint*, arXiv:2510.25726.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianfu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han-Chung Lee. 2026. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *CoRR*, abs/2602.12670.

Xiaoxiao Li. 2026. When single-agent with skills replace multi-agent systems and when they fail. *CoRR*, abs/2601.04748.

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

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Weinan E, and Siheng Chen. 2025. Ml-master: Towards ai-for-ai via integration of exploration and reasoning. *CoRR*, abs/2506.16499.

Shinji Mai, Yunpeng Zhai, Ziqian Chen, Cheng Chen, Anni Zou, Shuchang Tao, Zhaoyang Liu, and Bolin Ding. 2025. Cues: A curiosity-driven and environment-grounded synthesis framework for agentic rl. *Preprint*, arXiv:2512.01311.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. *Preprint*, arXiv:2311.12983.

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. *Preprint*, arXiv:2603.25158.

Alexander Novikov, Ngân Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. 2025. Alphaevolve: A coding agent for scientific and algorithmic discovery. *CoRR*, abs/2506.13131.

OpenAI. 2025. System Card for o3-mini. Accessed on December 11, 2025.

Yixin Ou, Yujie Luo, Jingsheng Zheng, Lanning Wei, Shuofei Qiao, Jintian Zhang, Da Zheng, Huajun Chen, and Ningyu Zhang. 2025. Automind: Adaptive knowledgeable agent for automated data science. *CoRR*, abs/2506.10974.

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. 2025. Reasoningbank: Scaling agent self-evolving with reasoning memory. *CoRR*, abs/2509.25140.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Shuofei Qiao, Yanqiu Zhao, Zhisong Qiu, Xiaobin Wang, Jintian Zhang, Zhao Bin, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2025. Scaling generalist data-analytic agents. *CoRR*, abs/2509.25084.

Ram Ramrakhya, Andrew Szot, Omar Attia, Yuhao Yang, Anh Nguyen, Bogdan Mazoure, Zhe Gan, Harsh Agrawal, and Alexander Toshev. 2025. Scaling synthetic task generation for agents via exploration. *CoRR*, abs/2509.25047.

Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, Jian Yang, Ge Zhang, Jiaheng Liu, Changwang Zhang, Jun Wang, Yuchen Eleanor Jiang, and Wangchunshu Zhou. 2025. Taskcraft: Automated generation of agentic tasks. *CoRR*, abs/2506.10055.

David Silver Sutton, Richard S. 2025. Welcome to the Era of Experience.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, et al. 2025. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*.

5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. 2025a. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *Preprint*, arXiv:2508.06471.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. 2025b. Kimi k2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 16022–16076. Association for Computational Linguistics.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and

Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2024.

Jiayu Wang, Yifei Ming, Zixuan Ke, Shafiq Joty, Aws Albarghouthi, and Frederic Sala. 2026a. Skillorchestra: Learning to route agents via skill transfer. *CoRR*, abs/2602.19672.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. 2025a. Reinforcement learning for self-improving agent with skill library. *CoRR*, abs/2512.17102.

Qihao Wang, Ziming Cheng, Shuo Zhang, Fan Liu, Rui Xu, Heng Lian, Kunyi Wang, Xiaoming Yu, Jianghao Yin, Sen Hu, Yue Hu, Shaolei Zhang, Yanbing Liu, Ronghao Chen, and Huacan Wang. 2026b. Memgovern: Enhancing code agents through learning from governed human experiences. *Preprint*, arXiv:2601.06789.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. 2025b. Inducing programmatic skills for agentic tasks. *CoRR*, abs/2504.06821.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025c. Agent workflow memory. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*. OpenReview.net.

Peng Xia, Jianwen Chen, Xinyu Yang, Haoqin Tu, Jiaqi Liu, Kaiwen Xiong, Siwei Han, Shi Qiu, Haonian Ji, Yuyin Zhou, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. 2026. Metaclaw: Just talk – an agent that meta-learns and evolves in the wild. *Preprint*, arXiv:2603.17187.

Hanwen Xing, Haomin Zhuang, Xuandong Zhao, Yue Huang, Zhenheng Tang, and Xiangliang Zhang. 2026. Recipes for agents: Understanding skills and their open questions. *Preprint, ResearchGate. doi*, 10.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *Preprint*, arXiv:2502.12110.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Chengyuan Yang, Zequn Sun, Wei Wei, and Wei Hu. 2026a. Beyond static summarization: Proactive memory extraction for llm agents. *Preprint*, arXiv:2601.04463.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. 2026b. Autoskill: Experience-driven lifelong learning via skill self-evolution. *Preprint*, arXiv:2603.01145.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2023. Webshop: Towards scalable real-world web interaction with grounded language agents. *Preprint*, arXiv:2207.01206.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *Preprint*, arXiv:2406.12045.

Yunzhi Yao, Jiaxin Qin, Ningyu Zhang, Haoming Xu, Yuqi Zhu, Zeping Yu, Mengru Wang, Yuqi Tang, Jia-Chen Gu, Shumin Deng, Nanyun Peng, and Huajun Chen. 2025. Rethinking knowledge editing in reasoning era. *Authorea Preprints*.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *Preprint*, arXiv:2504.13837.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic "differentiation" via text. *Preprint*, arXiv:2406.07496.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, Zhaoyang Liu, Bolin Ding, and Jingren Zhou. 2025. Agentevolver: Towards efficient self-evolving agent system. *Preprint*, arXiv:2511.10395.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. 2025a. G-memory: Tracing hierarchical memory for multi-agent systems. *Preprint*, arXiv:2506.07398.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. 2025b. Memevolve: Meta-evolution of agent memory systems. *Preprint*, arXiv:2512.18746.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. 2026a. Coevoskills: Self-evolving agent skills via co-evolutionary verification. *Preprint*, arXiv:2604.01687.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. 2026b. Memskill: Learning and evolving

memory skills for self-evolving agents. *CoRR*, abs/2602.02474.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2025c. Agentic context engineering: Evolving contexts for self-improving language models. *Preprint*, arXiv:2510.04618.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: LLM agents are experiential learners. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 19632–19642. AAAI Press.

Da Zheng, Lun Du, Junwei Su, Yuchen Tian, Yuqi Zhu, Jintian Zhang, Lanning Wei, Ningyu Zhang, and Huajun Chen. 2025. Knowledge augmented complex problem solving with large language models: A survey. *CoRR*, abs/2505.03418.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

YanZhao Zheng, ZhenTao Zhang, Chao Ma, YuanQiang Yu, JiHuai Zhu, Yong Wu, Tianze Xu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, and Gang Yu. 2026. Skillrouter: Skill routing for llm agents at scale. *Preprint*, arXiv:2603.22455.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, and Jun Wang. 2025. Memento: Fine-tuning llm agents without fine-tuning llms. *Preprint*, arXiv:2508.16153.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, Runyu Yang, Qiangbin Liu, Xinlei Yu, Jianmin Zhou, Na Wang, Chunyang Sun, and Jun Wang. 2026a. Memento-skills: Let agents design agents. *Preprint*, arXiv:2603.18743.

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. 2024. Webarena: A realistic web environment for building autonomous agents. *Preprint*, arXiv:2307.13854.

Tianyi Zhou, Dongrui Liu, Leitao Yuan, Jing Shao, and Xia Hu. 2026b. Colleague.skill: Automated ai skill generation via expert knowledge distillation.

A Detailed Experiments Settings

A.1 Benchmark Details

BFCL-v3 Berkeley Function Calling Leaderboard V3 (BFCL-v3) (Patil et al., 2025) is a benchmark for evaluating function calling and tool use in large language models. It emphasizes multi-turn interaction and multi-step reasoning. The benchmark contains over 1,800 test instances and supports multiple programming languages, including Python, Java, and JavaScript. Models are required to generate valid API calls and handle non-trivial interaction patterns. Evaluation considers both structural validity and functional correctness. We first check whether the generated code is syntactically valid using Abstract Syntax Tree analysis, and then execute it to verify that the outputs match the expected results. A task is considered successful only when the agent produces all required function calls with correct syntax and returns the correct computational outcomes. In this work, we report Avg@4, which measures the average task success rate across four independent trials, and Pass@4, which measures the probability that at least one of the four trials succeeds.

Appworld AppWorld (Trivedi et al., 2024) is a benchmark suite for evaluating function calling agents and interactive coding systems in realistic application environments. It simulates an ecosystem of nine widely used applications, such as email services, music streaming platforms, and payment systems, and provides 457 API endpoints together with activity data from around 100 virtual users. Tasks in AppWorld are typically long-horizon and require executing extended sequences of interdependent actions. Many tasks involve discovering appropriate APIs rather than directly reusing familiar patterns, which places additional demands on exploration and planning. The benchmark also exhibits a noticeable distribution gap between training and test sets, where API usage patterns and task structures in the test set differ from those observed during training. In addition, task execution is tightly coupled with the evolving environment state. Intermediate actions modify the system state and influence future decisions, which increases sensitivity to planning errors and makes robust multi-step reasoning more difficult. Evaluation is based on state-driven unit tests that assess task completion from multiple aspects. AppWorld provides both task-level and scenario-level metrics. In this work, we use Task Goal Completion as the primary measure of performance. Following the standard protocol, we report Avg@4 and Pass@4 across four independent trials.

τ^2-Bench τ^2-Bench (Barres et al., 2025) evaluates tool use in conversational agent settings, with a strong emphasis on user-agent interaction. The benchmark simulates multi-turn dialogues between a user and an agent, aiming to reflect realistic conversational behavior. Agents must track dialogue context across turns, interpret user requests, select and invoke APIs appropriately, and follow domain-specific business rules. The tasks cover domains such as airline customer service and retail customer service. The interactive nature of the benchmark requires agents to respond to user feedback, maintain coherent dialogue flow, and coordinate tool use with the ongoing conversation. Performance is assessed based on task completion accuracy, correctness of tool use, and compliance with policies. In this work, we conduct four independent trials per task and report Pass@1 on each of the three domains.

A.2 Baseline Details

No-memory No-memory performs inference without retrieving any prior experience.

A-Mem A-Mem (Xu et al., 2025) is an agentic memory framework that equips LLM-based agents with the ability to maintain and utilize long-term knowledge over extended interactions. The method organizes accumulated experiences into a memory-centric structure, enabling agents to selectively retain, retrieve, and revise stored information according to task objectives and observed outcomes. Rather than treating memory as a passive log, A-Mem emphasizes autonomous memory management driven by the agent's goals and interaction context. In our experiments, we reproduce A-Mem based on its publicly available implementation, with minor prompt adaptations to support memory writing and organization during task interactions.

AWM AWM (Agent Workflow Memory) (Wang et al., 2025c) is a memory-augmented agent framework that focuses on discovering reusable workflow patterns from past task executions. The method stores completed task trajectories as episodic experiences and derives higher-level procedural knowledge by analyzing multiple successful examples.

Experience retrieval follows a lightweight lexical matching strategy. Textual representations of task queries and stored experiences are mapped to sparse term-based vectors, and relevance is measured using cosine similarity. A small set of highly relevant experiences is selected for downstream analysis, with subsampling applied when multiple candidates exhibit comparable similarity. Workflow induction is performed by prompting a language model to analyze the retrieved successful trajectories and summarize recurring action patterns. Rather than relying on explicit symbolic rules or predefined workflow schemas, AWM captures reusable procedural structures directly from empirical task executions. Retrieved experiences are incorporated as conversational message objects (e.g., `HumanMessage` and `AIMessage`), enabling the language model to process exemplar interactions naturally within the dialogue context.

ExpeL ExpeL (Zhao et al., 2024) is an experience-driven learning framework that improves agent performance by reflecting on past successes and failures. The method stores task execution trajectories and generates experiential knowledge by contrasting successful and unsuccessful outcomes for the same task. In our experiments, we reproduce ExpeL by collecting both successful trajectories (reward ≥ 1.0) and failed trajectories (reward < 1.0). For each successful example, a small number of failed trajectories from the same task type are selected for comparative analysis. A large language model is prompted to analyze the paired trajectories and generate natural-language critiques that highlight key decision differences and improvement suggestions. These critiques are retained as unstructured textual experiences and reused as guidance in subsequent tasks.

A.3 Implementation Details

Skills Extraction. During the experience extraction stage, which comprises both reasoning and experience extraction, we employ GLM-4.6 with a temperature of 0.9. For each task in the training set, we independently sample four trajectories. Environment feedback exceeding 1500 tokens is summarized. We cluster the extracted skills using DBSCAN (Density-Based Spatial Clustering of Applications with Noise) with a cosine similarity threshold of 0.9. For each cluster, we truncate the skill set to at most 15 skills. Skill updates are performed with up to three iterative refinement rounds. During the skill expansion stage, we set the exploration model temperature to 1.0 and perform 1 time to explore environment for each training task.

Skills Usage. We build a skill semantic vector store using FAISS with an HNSW index under cosine similarity (via L2-normalized embeddings and inner-product search). At query time, we first perform a broad retrieval of the Top-100 nearest skills. Candidates are then filtered by a hybrid relevance threshold: we keep only results whose cosine similarity is at least 0.45, and also within 0.08 of the best match for that query, ensuring both a minimum quality floor and adaptive selectivity. To reduce near-duplicate skills, we apply semantic deduplication by removing items whose pairwise cosine similarity exceeds 0.95, retaining the higher-scoring representative. Finally, we return up to 8 skills after applying Maximal Marginal Relevance (MMR) for diversity-aware selection, using a relevance–diversity trade-off weight of 0.75 to emphasize relevance while mitigating redundancy.

B Further Analysis

B.1 `SkillX` can Expand Base Model's Capability Boundary.

We observe that experience-based learning leads to substantial Pass@4 improvements for the weaker models, K2 and Qwen3-32B. This suggests that, in practice, the most direct way to extend the capability boundary of a base model is to distill knowledge from a stronger model (Yue et al., 2025). In contrast, for the stronger model GLM-4.6, neither the baseline nor `SkillX` yields a significant gain in Pass@4. This indicates that stronger models already possess robust capabilities in exploration, planning, and tool use, leaving limited headroom for further capability expansion via experience-based augmentation. Nevertheless, the modest improvements still support the effectiveness of `SkillX`.

B.2 Evaluating `SkillX` Across Other Base Models

We further evaluate `SkillX` on stronger base models, including DeepSeek-V3.2 and GPT-4.1, which are at least comparable to, and in some cases stronger than GLM-4.6. We find that `SkillX` provides consistent performance gains, whether the skills are extracted by these stronger models themselves or constructed using GLM-4.6.

Methods	BFCL-v3		Appworld	
	Avg@4	Pass@4	Avg@4	Pass@4
DeepSeek-V3.2				
No Memory	64.33	81.33	61.90	84.08
SkillX				
GLM-Extract	67.17	83.33	64.28	86.90
Self-Extract	**67.83**	**84.67**	**65.48**	**88.39**
GPT-4.1				
No Memory	49.66	58.39	66.37	82.74
SkillX				
GLM-Extract	**60.00**	**69.33**	66.82	**84.52**
Self-Extract	50.67	56.67	**68.60**	82.14

Table 2: Performance of SkillX on other base models.

B.3 Ablation Study on Three Components of SkillX

We conduct ablation studies on the three key components of SkillX, i.e., *multi-level skills design*, *skills refinement*, and *skills expansion*, as shown in Table 3. The results in Table 3 suggest that SkillX is robust to its underlying experience representation, while iterative refinement and skill expansion can offer further improvements depending on the model and the particular combination of components.

Please note that we do not perform ablations of skills iteration and skills expansion on τ^2-Bench. This is because τ^2-Bench is a user-interactive benchmark whose tool schemas are relatively simple in both number and dependency structure, and its training set already covers many task patterns directly. More broadly, for user-centric benchmarks of this type (e.g., dialogue benchmarks), it remains an open question whether experience learning centered around tool-schema-based skills is the most appropriate formulation. Therefore, we believe that component studies on skill iteration and skill expansion are less suitable for τ^2-Bench, and we do not include them in our ablation experiments.

Model	Methods	BFCL-V3		AppWorld	
		Avg@4	Pass@4	Avg@4	Pass@4
GLM-4.6	No Memory	76.67	83.33	60.27	83.33
	Vanilla-Iter1	78.50	85.33	62.35	83.33
	Vanilla-Iter2	**79.50**	**86.00**	64.29	85.12
	Vanilla-Iter3	78.83	84.67	61.46	85.71
	Expand-Iter1	78.50	85.33	64.58	83.93
	Expand-Iter2	78.83	85.33	64.88	87.50
	Expand-Iter3	78.83	84.67	**64.88**	**88.69**

Table 3: Ablation results of SkillX on three components. Specifically, Vanilla-Iter1 uses only the *multi-level skills design*; Vanilla-Iter2 and Vanilla-Iter3 additionally incorporate *skills refinement*; Expand-Iter1 uses the *multi-level skills design* together with *skills expansion*; Expand-Iter2 and Expand-Iter3 combine *multi-level skills design*, *skills refinement*, and *skills expansion*.

C Case Study For SkillX

We present case studies across three diverse benchmarks: AppWorld (Trivedi et al., 2024), BFCL (Patil et al., 2025), and τ^2-bench (Barres et al., 2025). These cases show that skill libraries help agents avoid common failures such as incorrect API call sequences, missing prerequisite checks, and the inability to handle conversational topic shifts. By framing domain knowledge as reusable skills, agents can complete complex multi-step tasks that the baseline method fails, reducing trial and error from multiple failed attempts to successful execution on the first attempt.

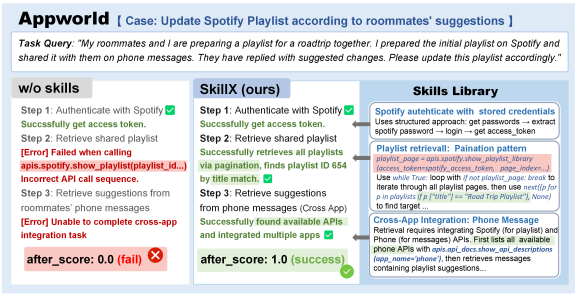


Figure 4: **AppWorld benchmark case study: Updating Spotify playlist based on roommates' suggestions.** SkillX successfully handles API call sequences (pagination pattern for playlist retrieval) and cross-app integration (integrating Spotify and Phone APIs), while the baseline without multi-level skills fails due to incorrect API call sequences and inability to complete cross-app integration tasks.

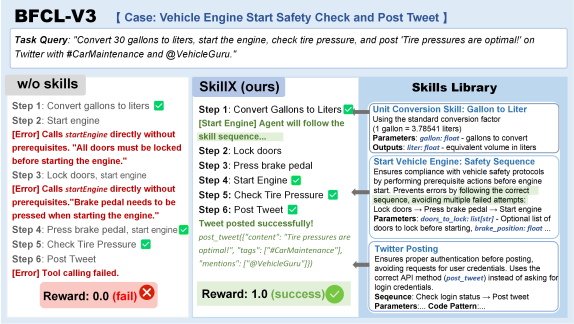


Figure 5: **BFCL benchmark case study: Vehicle engine start safety check and Twitter posting.** `SkillX` follows prerequisite sequences (lock doors → press brake pedal → start engine) and properly authenticates before posting tweets, while the baseline without multi-level skills fails by calling APIs without prerequisites and encountering tool calling errors.

Figure 6: **τ^2-bench case study: Requesting delay flight compensation in airline domain.** `SkillX` handles topic shifts, retrieves user reservations without reservation numbers, verifies flight delays, and executes the compensation workflow, while the baseline without multi-level skills fails to recognize topic shifts and cannot retrieve reservation details.

D Main Prompt Use For `SkillX`

In this section, we provide the prompts of `SkillX` used for skill extraction, planning, filtering, and merging operations.

D.1 General Filter Prompt

General Filter Prompt

You are a coding expert. Given a predefined skill, evaluate whether its quality is good or bad.

Evaluation guidelines:

1. **Domain specificity**: Check whether the skill includes domain-specific library names APIs, e.g., `{api}`.
2. **Over-encapsulation**: Check whether the skill's implementation merely calls a single other skill (i.e., it is just a thin wrapper).
3. **No-Python-libraries**: Check whether additional Python libraries are introduced in the skill.
4. **Reusability**: Check whether the parameters are specific.
5. **No-Functional style**: Check whether a functional style is being used (e.g., the presence of return).

Bad Example1
`{example}`

Bad Example2
`{example}`

Good Example
`{example}`

Only return "good" or "bad". Don't return any other words.

Table 4: Prompt for filtering skills based on quality criteria.

D.2 Tool Summary Prompt

Tool Summary Prompt

You are an AI assistant specialized in analyzing agent trajectories.
Your task is to summarize a single interaction: based on the environment feedback from the current step, extract and summarize the key information in no more than 50 words.

Inputs Description

1. The AI assistant's reasoning and action
2. The resulting environment feedback after the action

Summary Guidelines

1. Summarize what the environment feedback conveys in light of the AI assistant's intent.
2. Preserve details that are tightly relevant to the intent verbatim when possible; compress other redundant information.
3. Summarize only factual content from the environment feedback—do not invent anything.
4. Write the summary in the tone of the environment feedback.

Output Format
<feedback>
Your summary of the environment feedback
</feedback>

Table 5: Prompt for summarizing environment feedback from agent interactions.

D.3 Tool Schema Filter Prompt

Tool Schema Filter Prompt

You are a tool-invocation expert. Based on the tool specifications, verify whether the provided tool invocations are correct.

Input

1. **Tool invocation content**: may include one or multiple tool calls.
2. **Tool specifications**: including tool description, parameters, return schema, and other usage notes.

Judging Guidelines

1. **Parameter validation**: Check whether the invocation parameters comply with the specifications (e.g., missing required parameters, unsupported/nonexistent parameters, wrong types or formats, invalid values, etc.).
2. **Call dependency**: For multiple tool calls, verify that their order does not violate logical dependencies. If there is no dependency between the calls, ignore this check.
3. **Comment–function alignment**: Ensure the logic described in any comments matches what the tool is designed to do.
4. **Output Format**: Provide your reasoning and conclude with either 'correct' or 'fail', wrapped in <answer></answer>.

Table 6: Prompt for validating tool invocations against specifications.

D.4 Plan Extract Prompt

Plan Extract Prompt

You are a **Planning Expert**.
Your job is to analyze an agent's API interaction history and the user's task, then distill them into a concise, reusable plan. This plan should serve as a reference for handling similar tasks more effectively in the future.

OBJECTIVES

1. **Understand Capabilities**
   - Analyze the recorded API calls to identify the actual functional capabilities demonstrated.
2. **Abstract into a Plan**
   - For each feasible task supported by those capabilities, produce a concise, reusable step-by-step plan that can be applied to similar tasks.

Planning Creation Rules

1. Focus
- Do not simply restate each API function step-by-step using technical jargon. Instead, describe the underlying sub-goal behind each action segment.

2. Remove Non-Essential Steps
- Exclude capability exploration, debugging, and failed steps.

3. Reusability
- The plan must be precise enough for other models to reuse.

4. Conciseness
- Merge steps from the interaction history that share the same objective into a single sub-step in the plan.
- Use a compact writing style for each sub-step, while listing the key APIs involved in that step (one or more).
- Do not omit any critical, potentially required API keys.

OUTPUT FORMAT

For each task, output exactly one plan and follow this format strictly:
<plan>
step 1: A natural, specific, concise sub-task goal; key APIs used (one or more).
step 2: ...
...
</plan>

GOOD EXAMPLES

`{examples}`

CHECKLIST BEFORE FINALIZING

✓ **Reusability** — Ensure no critical steps are missing, and the step order is correct.
✓ **Conciseness** — Confirm there are no redundant or unnecessary steps.
✓ **Agent-centered** — Make sure the plan reads like actionable instructions that other models can reliably follow.

Table 7: Prompt for extracting reusable plans from agent trajectories.

D.5 Merge Prompt

Merge Prompt

You are a code expert. Your task is to analyze a list of skills, merge skills that are meaningfully similar, and decompose complex skills into smaller atomic skills while preserving behavior and intent.

Input Description
The user will provide a list of skills.

Skill Definition Rule

- Skill is a dictionary with four keys: `name`, `document`, `content` and `tools`.
  1. `name`: the skill's name.
  2. `document`: the skill's functionality, the key parameters, the final output of the skill, and any important notes.
  3. `content`: the concrete implementation of the skill.
  4. `tools`: the key tools used in the skill (list).
- The skill is abstract, modular, and reusable. Specifically, the skill name must be generic under one application (e.g., `{good example}` instead of `{bad example}`. The skill must use parameters instead of hard-coded values (e.g., specific email address `{email address}`). The skill body must be self-contained.
- Explicitly declare the key parameters and the final output data types using type hints. Example: `Parameters: param: str; Outputs: output: list[dict]:`
- Include a detailed description of the skill with input and output explanation.
- The skill should not be similar to the existing skills in the skills library.
- The skill must involve multiple processing steps. Simply using the result of an API call without additional logic does not qualify as a valid skill.
- Never call other skills from the skills library or any previously defined skills.
- Do not import any Python packages.
- Avoid a functional style; there's no need to use return.

Good skill:

```json
{
    "name": {name},
    "document": {document},
    "content": {content},
    "tools": {tools}
}
```

Focus

1. Focus on skills with similar names and similar skillality.
2. Carefully analyze the concrete implementation differences between similar skills.

Merge Guidelines

1. **Generality**: Merge skills that have similar names and similar skillality. The merged skill should use a generic name, and its **Notes** and implementation should cover all plausible variants and edge cases.
2. **Atomicity**: If skills have a containment relationship (one skill's skillality subsumes or builds on another), follow the skill definitions to preserve atomicity and avoid merging.
3. **Merge Constraints**: Any merged skill must comply with the skill definition rules, especially atomicity and reusability, and should avoid being tied to a specific task or scenario.

Decompose Guidelines

1. **Atomicity**: Only decompose skills whose skillality is overly complex (e.g., they include skillality already covered by other provided skills) into smaller sub-skills.
2. **Generality**: The decomposed skills must follow the skill-definition rules and remain reusable—avoid coupling them to any specific task or scenario.

Output Format
Output a list containing the skills (with one or multiple skills) from merging and/or decomposing the skills in the input skill list as follows:
<skill>
[
    "skill 1",
    ...
]
</skill>
Note: You don't necessarily need to both merge and decompose. You may choose to only merge them into a single skill.

Table 8: Prompt for merging and decomposing skills.

D.6 Atomic Skill Extract Prompt

Atomic Skill Extract Prompt

An agent system is provided with a **skills library** and has tried to solve the task multiple times with a successful solution. Review the task-solving attempt and extract generalizable skills.

1. Inputs Description

- **User Task**
- **Trajectory**: A record of an agent's interactions successfully with the environment as it attempts to complete a user task.
- **skills library**: A collection of all currently available skills that can be directly reused.
- **Specific-Tool**: Given a specific tool, extract only one reusable skill for the specified tool.

2. Skill Definition Rule

- Skill is a dictionary with four keys: `name`, `document`, `content` and `tools`.
  1. `name`: the specific tool's name.
  2. `document`: the tool's functionality, the key parameters, the final output of the skill, and any important notes.
  3. `content`: the tool's usage examples, and examples of combining it with other tools (if applicable).
  4. `tools`: the key tools used in the `content` (list).
- The skill is centered around a specific tool, describing its core functionality, important notes, and common usage examples.
- Explicitly declare the key parameters and the final output data types using type hints. Example: `Parameters: param: str; Outputs: output: dict:`
- Include a detailed description of the skill with input and output explanation.
- The skill should not be similar to the existing skills in the skills library.
- The parameters used in `content` must be reusable instead of hard-coded values (e.g., specific email address "jay@gmail.com")
- The usage examples of `content` may involve one or more tool uses.
- The `document` must clearly and thoroughly document all relevant details of the specific tool use.
- Never call other skills from the skills library or any previously defined skills.
- Do not import any Python packages.
- Avoid a functional style and Python code style; there's no need to use return.

3. Update Existing Skills

Your goal is to ensure the system retains actionable skills that help it behave correctly in the future.
You have three options: **[modify, add, keep]**

- **modify**: revise an existing skill to make it more effective (e.g., improving documents). Only change `content` when necessary, and ensure the resulting skill remains broadly general-purpose.
- **add**: introduce a new skill only when the existing skills library is missing the specified tool.
- **keep**: Preserve the skill unchanged when there are no clear issues.

Common actions:

- add a new skill
- update a skill's usage instructions/documentation
- revise a skill's variable/parameter definitions to make it more generalizable
- keep a skill unchanged

4. Requirements for each skill that is modified or added.

- **Avoid duplication**: If a skills library is provided, do not add new skills that are similar to existing ones—use **keep** or **modify** instead.
- **Ensure domain specificity**: The skill must contain domain-specific tool.
- **Specific-Tool guided extraction**: Only focus on the specified tool in the trajectory when extracting skills.

5. Good Skill Example

`{example}`

6. Output Format

You will finish by returning in this JSON format as follows:

```json
[
  {
    "option": "modify",
    "skill": "the modified skill",
    "modified_from": "spotify get all user playlists"
# specify the skill name of existing skills that is modified
  },
  {
    "option": "add",
    "skill": "the added skill",
  },
  {
    "option": "keep",
    "skill_name": "the kept skill name",
  }, ...
]
```

Note that your updated skills may not need to cover all the options. You can only use one type of updates or choose to remain all skills unchanged.

7. CHECKLIST BEFORE FINALIZING

- ✓ **Reusability** — Ensure no critical steps are missing, each skill is modular, all parameters are abstract rather than specific.
- ✓ **Optimality** — Ensure each skill meets the required definition standards.
- ✓ **Agent-centered** — Add helpful notes in each skill to guide other models in using it correctly.
- ✓ **Specific-Tool focus** — Whether the extracted skill doesn't center around this Tool?

Table 9: Prompt for atomic skill extraction based on specific tools.

D.7 Functional Skill Extract Prompt

Functional Skill Extract Prompt

An agent system is provided with a **skills library** and has tried to solve the task multiple times with a successful solution. Review the task-solving attempt and extract generalizable skills.

1. Inputs Description

- **User Task**
- **Trajectory**: A record of an agent's interactions successfully with the environment as it attempts to complete a user task.
- **skills library**: A collection of all currently available skills that can be directly reused.
- **Specific-step**: Given a concrete step, extract only one reusable skill for the specified step.

2. Skill Definition Rule

- Skill is a dictionary with four keys: `name`, `document`, `content` and `tools`.
  1. `name`: the skill's name.
  2. `document`: the skill's functionality, the key parameters, the final output of the skill and any important notes.
  3. `content`: the concrete implementation of the skill.
  4. `tools`: the key tools used in the skill (list).
- The skill is abstract, modular, and reusable. Specifically, the skill name must be generic under one application (e.g., `spotify get songs by genre` instead of `get pop songs`). The skill must use parameters instead of hard-coded values (e.g., specific email address "jay@gmail.com"). The skill body must be self-contained.
- Explicitly declare the key parameters and the final output data types using type hints. Example: `Parameters: param: str; Outputs: output: list[dict]`:
- Include a detailed description of the skill with input and output explanation.
- The skill should not be similar to the existing skills in the skills library.
- The skill must involve multiple processing steps. Simply using the result of an API call without additional logic does not qualify as a valid skill.
- Never call other skills from the skills library or any previously defined skills.
- Do not import any Python packages.
- Avoid a functional style; there's no need to use return.

3. Update Existing Skills

Your goal is to ensure the system retains actionable skills that help it behave correctly in the future.
You have three options: **[modify, add, keep]**

- **modify**: revise an existing skill to make it more effective (e.g., improving documents). Only change `content` when necessary, and ensure the resulting skill remains broadly reusable/general-purpose.
- **add**: introduce a new skill only when existing skills cannot support a critical step, in order to improve future performance.
- **keep**: Preserve the skill unchanged when there are no clear issues.

Common actions:

- add a new skill
- update a skill's usage instructions/documentation
- revise a skill's variable/parameter definitions to make it more generalizable
- if a skill is overly complex, refactor it into more modular skills (involving both **modify** and **add**)
- keep a skill unchanged

4. Requirements for each skill that is modified or added.

- **Avoid duplication**: If a skills library is provided, do not add new skills that are similar to existing ones—use **keep** or **modify** instead.
- **Exclude non-solution behavior**: Do not include capability exploration, debugging activities, or any failed/incorrect steps.
- **Ensure domain specificity**: The skill must reference domain-specific libraries/APIs, e.g., `{api}`.
- **Avoid over-wrapping**: Verify the implementation is not merely a thin wrapper around another skill (i.e., not just calling a single underlying skill without meaningful additional logic).
- **Specific-step guided extraction**: Only focus on the specified step in the trajectory when extracting skills.

5. Good Skill Example

`{example}`

6. Output Format

You will finish by returning in this JSON format as follows:

```json
[
  {
    "option": "modify",
    "skill": "the modified skill",
    "modified_from": "spotify get all user playlists"
# specify the skill name of existing skills that is modified
  },
  {
    "option": "add",
    "skill": "the added skill",
  },
  {
    "option": "keep",
    "skill_name": "the kept skill name",
  }, ...
]
```

Note that your updated skills may not need to cover all the options. You can only use one type of updates or choose to remain all skills unchanged.

7. CHECKLIST BEFORE FINALIZING

- ✓ **Reusability** — Ensure no critical steps are missing, each skill is modular, all parameters are abstract rather than specific.
- ✓ **Optimality** — Ensure each skill meets the required definition standards.
- ✓ **Agent-centered** — Add helpful notes in each skill to guide other models in using it correctly.
- ✓ **Specific-step focus** — Whether the extracted skill includes any content that does not belong to this step?

Table 10: Prompt for functional skill extraction based on specific steps.