# BEYOND SKILL EVOLUTION: NAVIGATING THE ADAPTATION PATH FOR SKILL LIBRARIES

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

A *skill* is a structured natural-language document that integrates tool descriptions and workflow examples, and can be reused across tasks. However, the performance of the same Large Language Model (LLM) agent models varies a lot on the same skill library. Existing reflective skill evolution methods answer only *what* to evolve: they edit contents along a single adaptation trajectory for a single skill, lacking an explicit mechanism for locating failures to evolve across a multi-skill library, and failing to break away from the evolution paths with accumulated suboptimal edits. In this paper, we reframe skill evolution as an agentic tree search problem over the two challenges and propose **SkillPilot**, a Meta-Optimizer-Guided Monte Carlo Evolution Tree Search (MCETS) framework that navigates where and which way to evolve for a skill library in four recurrent stages: selection, expansion, simulation, and backpropagation, flexibly navigating the evolution path, which prevents the agent from over-exploring the suboptimal evolution path. Extensive experiments have demonstrated SkillPilot's superior performance on agent-generated skills and human-curated skills on three benchmarks with various task types. Our code is available at https://anonymous.4open.science/r/SkillPilot-EF1C/.

## 1 INTRODUCTION

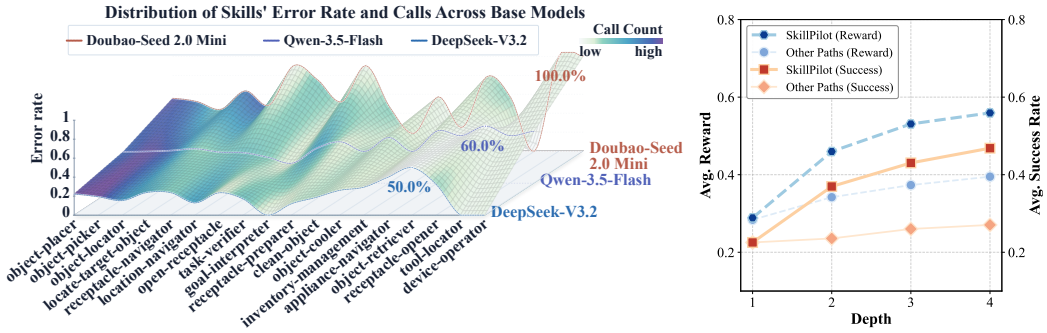
LLM agents are increasingly built around skills, structured natural-language documents that consist of descriptions, workflows, instructions for available tools (Xu & Yan, 2026; Zhou et al., 2026b), and examples to solve long-horizon tasks such as household manipulation (Shridhar et al., 2020), web shopping (Yao et al., 2022a; Yu et al., 2026a), or data analysis (Li et al., 2026b). Existing research has demonstrated that well-curated skills can boost an agent's success rate substantially and enhance the efficiency of the agents (Li et al., 2026b; Liang et al., 2026), as the reusable workflows can help the agent find solutions to repeated task patterns. Existing skills curation work mainly focuses on the sampled trajectories or sub-trajectories, summarizing them into multiple skill documents (Wang et al., 2023; Yang et al., 2026b; Liang et al., 2026; Lin et al., 2026; Hu et al., 2026; Ni et al., 2026).

Despite the substantial gains these skills deliver on long-horizon tasks, the same skill library rarely performs equally well and has different invocation distributions on different base models, as shown in Figure 1(a). In addition, changes in skill contents can substantially shift in-context learning behaviors (Dou et al., 2026), especially when the skill library has numerous skill documents and is structurally complex (Yang et al., 2026b; Liang et al., 2026; Wu et al., 2026). Therefore, a static skill document exhibits a clear gap with respect to in-context learning performance, and this gap should be closed by deeply **adapting the skills to the target model**.

To improve skill-use performance, early skills evolution methods are developed to optimize skill from failures iteratively by applying patches (Zhang et al., 2026a; Zheng et al., 2025; Zhang et al., 2026d). For example, some training-free methods such as PSN (Shi et al., 2026a) optimize skills in the graph of skill candidates. Moreover, Skill-Evolver (Zhang et al., 2026a) expands multiple nodes for each refinement, while SkillOpt optimizes skills on minibatches of trajectories (Yang et al., 2026a). In addition, RL-based methods like Skill-RL (Xia et al., 2026) and ASI (Wang et al., 2025b) discover and refine skills from large amounts of rollouts. However, all of these methods edit the skill along a *single adaptation trajectory*, leading to the following challenges: (1) **Accumulated errors or conflicts in a single evolution path**: Across a multi-skill library, for an error pattern



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




(a) Error rate and invocation distribution of agent skills in ALFWorld. (b) SkillPilot vs. Random Evolution.

Figure 1: (a) The invocation behaviors vary across different models. For example, DeepSeek-V3.2 controls the error rate under 50%, focusing on reusing head skills. (b) Evaluated in SkillsBench, randomly sampling 5 random evolution paths for each task suffer from sub-optimal evolution gains.

the agent may propose different refinement strategies with different performance. Trapping the evolution of multiple skills on a single path leads to accumulated errors or conflicts, making the high-quality skill refinement in the intermediate process underutilized. This phenomenon is shown in Figure 1(b), where randomly sampled evolution paths are usually suboptimal. These two gaps collectively underscore the need for a skill evolution paradigm that can jointly optimize multiple skill documents and flexibly navigate the evolution space. (2) **Failing to locate failures across a multi-skill library**. In real-world scenarios, each agent task usually requires multiple documents from different skills. Optimizing one skill document at a time can lead to low evolution efficiency and conflicts in highly relevant skills (Wei et al., 2026). Therefore, our research question is how to adaptively navigate the evolution path to jointly optimize the skill library on the target model.

In this paper, we focus on skill adaptation, which is a skill evolution on the target agent model. We propose **SkillPilot**, a meta-optimizer-guided Monte Carlo Evolution Tree Search (MCETS) framework that explicitly searches the evolution path for jointly optimizing multiple skills. We consider skills as trainable representations (Zhou et al., 2026b), and reformulate the skill evolution as a structured search problem with the goal of reducing the gap in skill usage. To address challenge (1), SkillPilot employs an MCTS strategy over the skill evolution space, which bridges the gap between usage patterns and skill content by overcoming error accumulation induced by single-path evolution, adaptively exploring skill evolution pathways tailored to the model's capabilities. To address challenge (2), we develop a hierarchical meta-optimizer in the expansion stage of MCETS. Through iterative clustering, the meta-optimizer extracts and decouples the errors, and attributes them to different invoked skills to locate the flaws across a multi-skill library. This guidance helps the agent jointly optimize multiple skill files with located patches in the evolution tree search. SkillPilot plays the role of a navigator for the target model, flexibly pointing the way toward the right skill evolution direction adapted for the in-context learning behavior of the agent model, efficiently exploring high-quality skill patches for the skill library, and substantially improving skill-use quality.

Our contributions are summarized as follows: (1) We introduce SkillPilot, a meta-optimizer-guided MCETS algorithm that performs deep tree-structured exploration, which is the first work to flexibly navigate the skill adaptation path. (2) We develop a hierarchical meta-optimizer in the expansion stage as guidance, which improves the exploration efficacy and efficiency in the huge space of evolution. (3) Evaluated across various base models and skill types, our tree-search-based evolution substantially improves execution accuracy and steps, demonstrating the effectiveness of SkillPilot.

## 2 Method

### 2.1 Problem Formulation

Let $\pi_\theta$ be a frozen LLM agent and $\mathcal{S} = \{S_j\}_{j=1}^{J}$ denote its skill library. Each skill in the library $S_j = [s^{(j,1)}, \ldots, s^{(j,n_j)}]$ is a $j$-th structured document with $n_j$ editable sections or code blocks. Given a task $x$, the agent invokes skills from $\mathcal{S}$ and produces an execution trajectory $\tau \sim \pi_\theta(\cdot \mid x, \mathcal{S})$,

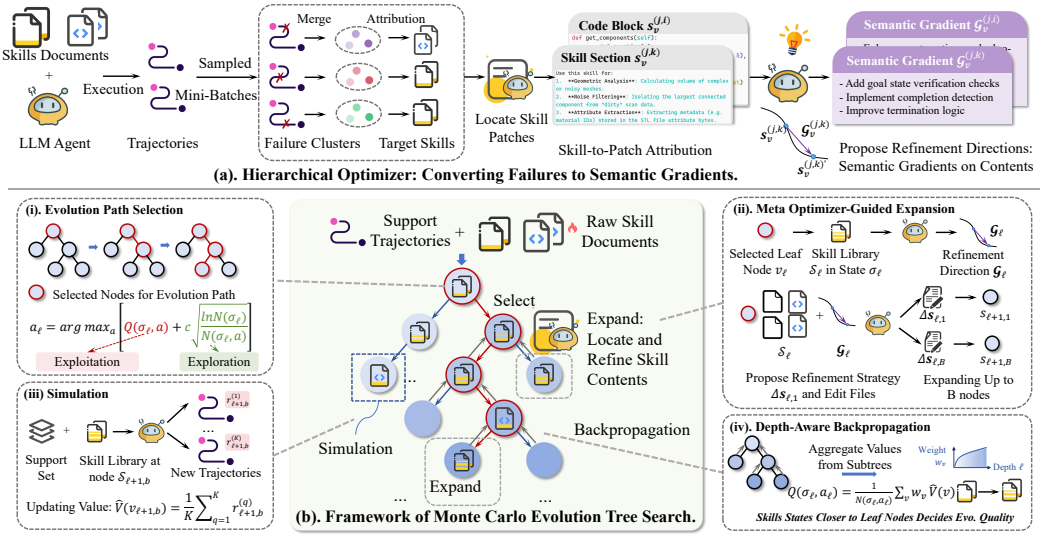



Figure 2: Overall Framework of SkillPilot. (a). The hierarchical meta-optimizer converts target-model failures into local semantic gradients for MCETS expansion. (b) Framework of MCETS. Path selection based on UCB1 flexibly selects the most valuable evolution path; after node expansion and simulation, the depth-aware backpropagation focuses on the values of deeper nodes.

from which the environment returns a reward $r(\tau) \in [0, 1]$. We regard the skill evolution as a meta-learning task, where the skills representation can be optimized in several steps to adapt to a new agent model. Starting from an initial library $\mathcal{S}_0$, with agent model weights $\theta$ fixed, we aim to find an optimized library that maximizes the expected reward over the task distribution $\mathcal{D}$:

$$\mathcal{S}^* = \arg\max_{\mathcal{S}'} \mathbb{E}_{x\sim\mathcal{D},\, \tau\sim\pi_\theta(\cdot|x,\mathcal{S}')}[r(\tau)]. \tag{1}$$

In practice, this expectation is estimated with a small support set $\mathcal{F}_0 = \{(x_k, \tau_k, r_k)\}_{k=1}^{K}$ consisting of failed trajectories. In this paper, we regard each skill representation $\mathcal{S}_j$ as a trainable object to enhance the execution performance of LLM agents by recursively refining the skills, and regard each section/code block as a multi-armed bandit in *agentic tree search* for the object in equation 1. We justify this idea in Appendix D. In the search tree, node $v$ stores a complete candidate library $\mathcal{S}_v$, and is edited by action $a = \{(j_t, i_t, o_t, \eta_t)\}_{t=1}^{m_a}$ that specifies a target skill $j$, a target block $i$, an edit operation $o$ and patch $\eta$ iteratively, where $m_a$ is the maximum number of edits for each mode. Consequently, the aim is to search for an optimal path (Yang et al., 2026a) that yields the skills for the problem (1). Uniquely, in the vast search space, SkillPilot is developed to decouple the failures to decide *which path* to evolve and locate the patches to jointly optimize multiple skills.

### 2.2 Hierarchical Meta-Optimizer: From Failures to Semantic Gradients

Directly editing from complete trajectories couples three difficult decisions: which failures share a root cause, which invoked skill is responsible, and which sections or code blocks should be refined (Alzubi et al., 2026). However, existing methods struggle to jointly attribute these failures to skill contents for multiple skills in large skill libraries. To address this challenge, SkillPilot assigns these decisions to a hierarchical meta-optimizer, which produces the directions of refinement for multiple skills as the semantic gradient and answers *where* the agent skills should jointly evolve, which is shown in Figure 2(a). The necessity of the hierarchical meta-optimizer in the vast search space of MCETS is analyzed in Appendix D.4, D.5, and D.6.

For a node $v$, let $\mathcal{E}_v = \{(x_k, \tau_k, r_k)\}_{k=1}^{K_v}$ be the trajectories generated with $\mathcal{S}_v$. We retain the trajectories that fail to finish the tasks or exhibit low rewards in $\mathcal{F}_v = \{(x_k, \tau_k, r_k) \in \mathcal{E}_v : r_k < \gamma\}$, where $\gamma$ is the task-specific success threshold. Each record includes the task, actions, observations, invoked skills, and the corresponding skill contents, so that a failure can be grounded in the contexts of skills invocation that the agent actually used.

**Iterative Failure-to-skill Attribution.** The meta-optimizer first disentangles the failures at the skill level iteratively. By repeatedly sampling a mini-batch records $(x, \tau, r)$ from the failure set $\mathcal{E}_v$ , it iteratively merges failures that are caused by similar reasons until all the failure records are read:

$$\mathcal{Z}_v^{(t+1)} \sim \pi_\theta^{\text{skill}}\Big(\cdot\big|\mathcal{S}_v, \mathcal{Z}_v^{(t)}\Big), \quad \mathcal{Z}_v^{(t)} = \{(C_m^{(t)}, \mathcal{A}_m^{(t)}, u_m^{(t)})\}_{m=1}^{M_v^{(t)}}, \tag{2}$$

where $C_m^{(t)} \subseteq \mathcal{F}_v$ is a failure cluster analyzed by policy $\pi_\theta^{\text{skill}}$ at the $t$-th merge round, $\mathcal{A}_m^{(t)} \subseteq \{1, \ldots, J\}$ is the set of skill indices attributed to $C_m^{(t)}$, $u_m^{(t)}$ is the root cause, and $M_v^{(t)}$ is the number of clusters. By merging all failed trajectories, we obtain the attribution $\mathcal{Z}_v = \{(C_m, \mathcal{A}_m, u_m)\}_{m=1}^{M_v}$.

**Skill-to-Patch Attribution.** Skill-level attribution is still not specific enough to guide a local edit for evolving a set of highly relative skills. To tackle this challenge, we attribute each failure cluster $C_m$ to the implicated skill $S_j$, and propose potential optimization directions for skill patches. First, the meta-optimizer performs a decomposition over the skill's sections or code blocks:

$$\mathcal{P}_v^{(m,j)} \sim \pi_\theta^{\text{patch}}\Big(\cdot|C_m, u_m, S_v^{(j)}, \mathcal{M}_v\Big) \subseteq \{1, \ldots, n_j\}, \qquad j \in \mathcal{A}_m, \tag{3}$$

where $\mathcal{P}_v^{(m,j)}$ contains the indices of patches for which the trajectories provide evidence of failures. $\mathcal{M}_v$ is the evolution memory, which enables the agent to focus on failures that are hard to solve in deeper explorations. For each targeted patch the meta-optimizer generates a local semantic gradient

$$\mathcal{G}_v^{(j,i,m)} \sim \pi_\theta^{\text{grad}}\Big(\cdot\big|s_v^{(j,i)}, C_m, u_m, S_v^{(j)}, \mathcal{M}_v\Big), \qquad j \in \mathcal{A}_m,\ i \in \mathcal{P}_{m,j}, \tag{4}$$

where $\mathcal{G}_v^{j,i,(m)}$ is a concise modification direction that specifies the diagnosed defect and the skill contents to be aligned with the agent. Different skill patches may be caused by the same failures , leading to potential overlaps or conflicts in local gradients. Let $\mathcal{X}_v^{(j,i)} = \{m : j \in \mathcal{A}_m,\ i \in \mathcal{P}_{m,j}\}$ denote the clusters that the sections $s^{(j,i)}$ attributes to. We aggregate their gradients as

$$\mathcal{G}_v^{(j,i)} = \pi_\theta^{\text{merge}}\Big(\cdot\big|s_v^{(j,i)}, \{(C_m, u_m, \mathcal{G}_v^{(j,i,m)})\}_{m\in\mathcal{X}_{j,i}}\Big) \quad \text{if } \mathcal{X}_v^{(j,i)} \neq \varnothing. \tag{5}$$

The agent policy $\pi_\theta^{\text{merge}}$ is a semantic merge, which resolves the conflicts among directions. Finally, we accumulate a memory within the MCETS, which guides the tree exploration, directing the agent to focus on finer-grained details and narrower scopes as it delves deeper into the failed trajectories.

Therefore, in the hierarchical meta-optimizer, the skill and patch attribution identify where adaptation is needed for multiple implicated skills. In addition, it also generates the semantic gradient $\mathcal{G}_v^{(j,i)}$ that specifies what should be modified to evolve the skills. SkillPilot uses the meta-optimizer's guidance as an expansion prior and decides how to compose the proposed local refinement through environment reward. Through this strategy, ***the search space of multiple skills to be evolved can be restricted in the semantic space of clustered failure groups.***

### 2.3 Meta-Optimizer-Guided Monte Carlo Evolution Tree Search

Single-path skill editing may accumulate an early mistake or discard a promising intermediate refinement, especially when edits interact across skills (Ji et al., 2026). We therefore use meta-optimizer-guided MCTS (Browne et al., 2012) to search over the skill evolution tree while balancing exploration and exploitation. The search consists of four stages: evolution-path selection, meta-optimizer-guided expansion, simulation, and back-propagation, which is shown in Figure 2(b).

**Evolution Path Selection.** We consider the node $v$ in any search trajectory of the tree at depth $\ell$. For simplicity, we denote the state of node $v$ as $\sigma_\ell = (\mathcal{S}_\ell, h_\ell, \mathcal{M}_\ell, \mathcal{G}_\ell)$, where $\mathcal{S}_\ell$ is the complete candidate library, $\mathcal{G}_\ell$ is the semantic gradient and $h_\ell$ is its edit history. The corresponding search trajectory is $\mathcal{H}_\ell = \big(\sigma_0, a_0, \sigma_1, \ldots, a_{\ell-1}, \sigma_\ell\big)$ for skills evolution, where $a_\ell$ denotes the edit action, and the tree is initialized with $\mathcal{T}_0 = \{\mathcal{H}_0\}$ where $\mathcal{H}_0 = (\sigma_0)$. At each depth, SkillPilot recursively selects the edge with the largest upper-confidence value

$$a_\ell = \arg\max_{a\in\mathcal{A}(\sigma_\ell)} \left[Q(\sigma_\ell, a) + c\sqrt{\frac{\ln N(\sigma_\ell)}{N(\sigma_\ell, a)}}\right], \tag{6}$$

where the first term $Q(\sigma_\ell, a)$ is the empirical value controlling exploitation of the edit edge $(\sigma_\ell, a)$, $N(\sigma_\ell)$ and $N(\sigma_\ell, a)$ are the state and edit-edge visit counts, $\mathcal{A}(\cdot)$ is the valid action space guided by meta-optimizer based on $\sigma_\ell$. If $N(\sigma_\ell, a) = 0$ the node is visited for exploration. This rule exploits search paths that perform well while retaining nodes whose values remain underexplored.

**Expansion guided by the meta-optimizer.** To fully bridge the gap between the skill library and the agent's in-context learning ability, we expand one child node based on the semantic instructions at the selected node. Specifically, for each parent node, up to $B$ refinement strategies can be applied throughout MCETS to obtain skill patches for the current node, generating up to $B$ child nodes.

$$\Delta\mathcal{S}_{\ell,b} \sim \pi_\theta^{\text{edit}}(\cdot|\mathcal{G}_\ell, \mathcal{S}_\ell), \quad \mathcal{S}_{\ell+1,b} \leftarrow \text{Edit}(\mathcal{S}_{\ell,b}, \Delta\mathcal{S}_{\ell,b}), \quad b = 1, \ldots, B, \tag{7}$$

where each proposal from the edit policy $\pi_\theta^{\text{edit}}$ explicitly generates a patch on the target section/code block grounded in semantic gradients, and $\mathcal{S}_{\ell+1,b}$ is the skill library of the expanded child node. In $b$-th child node, $\Delta\mathcal{S}_{\ell,b} = \{\eta_{\ell,b,1}, \cdots, \eta_{\ell,b,m_a}\}$ contains the patches $\eta$ for the skill content to be refined by the operation $o \in \{\text{modify}, \text{add}, \text{delete}, \text{create}\}$, and the process in Eq.(7) is defined as the action $a_{\ell,b}$. When an attributed failure spans coupled skills, it may contain a bounded set of co-dependent edits. Then we update the search trajectory and derive new tree branches:

$$\mathcal{H}_{\ell+1,b} \leftarrow \mathcal{H}_\ell \oplus (a_{\ell,b}, \sigma_{\ell+1,b}), \quad \mathcal{T}_{\ell+1} \leftarrow \mathcal{T}_\ell \cup \{\mathcal{H}_{\ell+1,b} | b \in \{1, \ldots, B\}\}, \tag{8}$$

where $\mathcal{T}_l$ denotes the search tree structure at the $l$-th depth.

**Simulation.** For each expanded candidate $v_{\ell+1,b}$, we execute the frozen agent on $K_{\text{sim}}$ support tasks and derive the execution trajectory-level reward of the newly expanded child node in each iteration:

$$\widehat{V}(v_{\ell+1,b}) = \frac{1}{K_{\text{sim}}} \sum_{q=1}^{K_{\text{sim}}} r\left(\tau_{\ell+1,b}^{(q)}\right), \quad \tau_{\ell+1,b}^{(q)} \sim \pi_\theta(\cdot \mid x_q, \mathcal{S}_{\ell+1,b}), \tag{9}$$

where $\{x_q\}_{q=1}^{K_{\text{sim}}}$ is a fixed set of failed support tasks used to estimate the value of each candidate node. Among $K_{\text{sim}}$ execution trajectories, the failed ones $\tau_{\ell+1,b}$ are processed by the hierarchical meta-optimizer to produce the semantic gradient before the node is expanded in a later iteration.

**Depth-Aware Backpropagation.** As the search trajectory gets deeper, the values of previous nodes in it are also updated for flexible evolution path navigation. Existing MCTS-based search methods applied to LLM agents (Wei et al., 2025; Zhang et al., 2026e; Gan et al., 2025) regard each node's contribution to value estimation of ancestor nodes as the same in a single path. However, this strategy *is no longer applicable to skills evolution*: the deeper nodes dominate the quality of the expansion in the next iteration. To address this challenge, SkillPilot employs depth-aware backpropagation to allocate higher weights to the deeper nodes, so that value estimates of nodes reflect high-quality leaves and the corresponding path can be prioritized in subsequent expansions.

The value of an edge aggregates all evaluated descendants in its subtree. Precisely, let $v$ be the leaf node on $\mathcal{H}_v$, and $\mathcal{V}(\sigma, a)$ be the multiset of evaluated nodes whose search paths contain edge $(\sigma, a)$:

$$Q(\sigma, a) = \frac{1}{N(\sigma, a)} \sum_{v' \in \mathcal{V}(\sigma,a)} w_{v'} \widehat{V}(v'), \quad w_{v'} = \frac{\text{idx}_{\mathcal{H}_v}(v')}{\mathcal{L}(\mathcal{H}_v)}, \tag{10}$$

where $N(\sigma, a) = |\mathcal{V}(\sigma, a)|$, $\mathcal{H}_v$ is the search path that contains leaf node $v$, $\text{idx}(v')_{\mathcal{H}_v}$ calculates the depth of $v'$ in $\mathcal{H}_v$ while $\mathcal{L}(\mathcal{H}_v)$ calculates the total depth. This design intentionally favors exploration of deeper paths with higher leaf values. Appendix E analyzes this weighting under a worst-case deceptive subtree, while Appendix D studies the benefit of exploiting section structure.

**Termination and Output.** The loop stops when the rollout budget is exhausted, the maximum depth is reached on all active paths, or no new valid edit can be generated. Finally, we derive a search tree $\mathcal{T}_L$ of the depth $L$ for skills adaptation, and the final skill library output is selected by maximum observed task reward since they exhibit high potential to address the failures:

$$v^* = \arg\max_{v \in \mathcal{N}_{\text{tree}}} \widehat{V}(v), \quad \mathcal{S}^* = \mathcal{S}_{v^*}, \tag{11}$$

where $\mathcal{N}_{\text{tree}}$ is the set of tree nodes of $\mathcal{T}_L$. At deployment, $\mathcal{S}^*$ is integrated into the original agent interface by loading only the skill patches, thereby significantly reducing storage overhead for real-world applications. With the MCETS algorithm, SkillPilot transcends mere exploration of what to evolve; it strategically selects optimal evolution paths, enabling deep and joint adaptation of skills to the target model. Algorithm 1 provides the complete procedure of our SkillPilot.

Table 1: Success rate (%) / average number of steps on ALFWorld and Task Completion Rate (%) / reward (%) / average number of steps on WebShop. Higher is better for SR, TCR, and Reward, and lower is better for Avg. Steps. The best results are **bold** and the second best are underlined. "*" indicates the significantly improvement over the baselines using two-sided t-test.

| | | ALFWorld Seen | | ALFWorld Unseen | | WebShop | | |
|---|---|---|---|---|---|---|---|---|
| **Base Model** | **Method** | **SR** | **Avg. Steps** | **SR** | **Avg. Steps** | **TCR** | **Reward** | **Avg. Steps** |
| DeepSeek V3.2 | ReAct | 62.86 | 20.23 | 67.91 | 19.52 | 45.81 | 23.65 | 23.73 |
| | Expel | 67.14 | 19.21 | 70.90 | 19.05 | 37.99 | 25.47 | 25.53 |
| | Skills | 80.71 | 15.53 | 80.60 | 15.98 | 61.45 | 40.20 | **20.48** |
| | SkillOpt | 83.57 | **13.98** | 86.57 | 13.63 | 58.10 | 39.42 | 20.77 |
| | Skill-Evolver | 82.85 | 14.82 | 85.07 | 14.88 | 54.75 | 38.72 | 21.89 |
| | **SkillPilot** | **87.86*** | 14.68 | **90.92*** | **13.42*** | **62.57*** | **43.37*** | 21.15 |
| Doubao-Seed 2.0 Mini | ReAct | 25.71 | 27.03 | 17.91 | 27.37 | 11.73 | 6.632 | 27.88 |
| | Expel | 30.71 | 24.91 | 20.15 | 27.17 | 22.91 | 13.05 | 26.28 |
| | Skills | 42.14 | 22.50 | 40.60 | 23.72 | 13.97 | 8.864 | 27.85 |
| | SkillOpt | 42.85 | 22.23 | 41.04 | 23.45 | 18.99 | 11.79 | 25.77 |
| | Skill-Evolver | 43.57 | 22.07 | 43.28 | 22.73 | 22.35 | 12.98 | 25.32 |
| | **SkillPilot** | **50.71*** | **20.97*** | **44.78*** | **21.95*** | **23.46*** | **15.14*** | **24.61*** |
| Qwen3.5-Flash | ReAct | 55.71 | 20.85 | 59.70 | 19.95 | 44.69 | 28.83 | 21.35 |
| | Expel | 63.57 | 20.23 | 61.19 | 21.59 | 41.90 | 25.97 | 22.00 |
| | Skills | 65.00 | 17.04 | 71.64 | 18.26 | 39.66 | 26.68 | 22.92 |
| | SkillOpt | 64.28 | 18.41 | 71.64 | 17.99 | 38.55 | 23.69 | 22.15 |
| | Skill-Evolver | **75.14** | **15.80** | 68.66 | 16.09 | 45.25 | 30.01 | **20.20** |
| | **SkillPilot** | 70.00 | 17.26 | **76.12*** | **16.19*** | **46.37*** | **31.26*** | 20.88 |

# 3 EXPERIMENTS

In this section, we evaluate SkillPilot on three benchmark datasets, including **ALFWorld** (Shridhar et al., 2020), **WebShop** (Yao et al., 2022a) and **SkillsBench** (Li et al., 2026b). The main results are discussed in Section 3.2 and 3.3, respectively. In Section 3.4, we analyze the Reusability of optimized skills among different subtasks, followed by the ablation study in Section 3.5.

## 3.1 EXPERIMENTAL SETUP

**Datasets and Evaluation Metrics.** In our experiment, we employ three benchmark datasets to verify the performance of skills evolution methods, including **ALFWorld** (Shridhar et al., 2020), **WebShop** (Yao et al., 2022a), and **SkillsBench** (Li et al., 2026b). On ALFWorld and WebShop, we report success rate (SR) and average number of steps. For WebShop, we report the task completion rate (TCR), reward, and average steps. On SkillsBench, we report the Reward and Pass Rate. The detailed introduction of these benchmarks and metrics can be found in Appendix B.1 and B.3.

**Baselines.** On ALFWorld and WebShop, we compare basic reasoning and skill-based baselines **ReAct** (Yao et al., 2022b), **Expel** (Zhao et al., 2024) and **Skills** (Liang et al., 2026), and skill evolution-based methods including **SkillOpt** (Yang et al., 2026a) and **Skill-Evolver** (Zhang et al., 2026a). On SkillsBench, we compare **Without Skills (ReAct)**, **With Skills** (provided by SkillsBench), and **Skill-Evolver** against our SkillPilot. The introduction of baselines, implementation details, and adopted prompts in the experiment can be found in Appendix B.2, B.4, and B.6.

## 3.2 OVERALL PERFORMANCE ON ALFWORLD AND WEBSHOP

In this subsection, we compare SkillPilot with baselines to evaluate its effectiveness on long-horizon interactive tasks. The results on ALFWorld and WebShop are shown in Table 1, which presents a comprehensive comparison of SkillPilot against state-of-the-art baselines across three backbones.

**SkillPilot delivers the strongest overall effectiveness in both environments.** By flexibly navigating the path to evolve, SkillPilot deeply and jointly optimizes skills, improving success rate while shortening trajectories in most cases. For instance, on DeepSeek-V3.2, it improves the success rate of ALFWorld Seen and Unseen tasks by 5.13% and 5.02%, respectively; and on Doubao-Seed 2.0 Mini, it improves the success rate of Seen and Unseen tasks by 16.39% and 3.47%, respectively, with

Table 2: Results of Average Reward (%), Pass Rate (%) and normalized gain $g$ (%) on SkillsBench, where Curated Skills are provided by the benchmark.The *Improv.* row reports SkillPilot's relative improvement over the best baseline. The best results are **bold**, and the second best are underlined. "*" indicates the significantly improvement over the baselines using two-sided t-test.

| | | Curated Skills (Trial=5) | | | Self-Generated Skills (Trial=3) | | |
|---|---|---|---|---|---|---|---|
| **Base Model** | **Method** | **Reward** | **Pass Rate** | $g$ | **Reward** | **Pass Rate** | $g$ |
| DeepSeek-V4-Flash | Without Skills | 8.760 | 2.590 | – | 8.760 | 2.590 | – |
| | Skills | 63.83 | 37.28 | 35.61 | 18.18 | 5.060 | 2.536 |
| | Skill-Evolver | 68.59 | 45.75 | 44.31 | 24.25 | 11.81 | 9.465 |
| | **SkillPilot** | **74.83*** | **54.57*** | **53.36*** | **36.01*** | **18.14*** | **15.96*** |
| | *Improv.* | +9.098% | +19.28% | +20.42% | +48.49% | +53.60% | +68.62% |
| GLM-5.3-Flash | Without Skills | 19.47 | 12.62 | – | 19.47 | 12.62 | – |
| | Skills | 28.38 | 22.53 | 11.34 | 24.73 | 17.70 | 5.814 |
| | Skill-Evolver | 42.18 | 31.24 | 21.31 | 37.17 | 30.86 | 20.87 |
| | **SkillPilot** | **54.81*** | **44.33*** | **36.29*** | **47.31*** | **38.27*** | **29.35*** |
| | *Improv.* | +29.94% | +41.90% | +70.30% | +27.28% | +24.01% | +40.63% |

the average number of steps reduced by around 1 step. Similar results can also be found in WebShop, where the reward and completion rate are also improved compared to strong baselines. We also perform a hyperparameter analysis in Section B.5 to verify the scaling behavior of SkillPilot.

**Skill library optimized by SkillPilot exhibits high transferability.** According to the results of Alfworld, SkillPilot substantially improves the success rate in DeepSeek-V3.2 and Doubao-Seed 2.0 Mini. For Qwen3.5-Flash, the performance of the skill evolution baselines, including SkillOpt and Skill-Evolver (71.64% and 68.66%), fails to outperform the basic skills (71.64%), which shows that optimized skills of these methods struggle to transfer to unseen tasks. By flexibly selecting the valuable paths to explore skill contents adapted to the target model, SkillPilot empowers the skill library with generalizable contents, enabling transferability to unseen tasks.

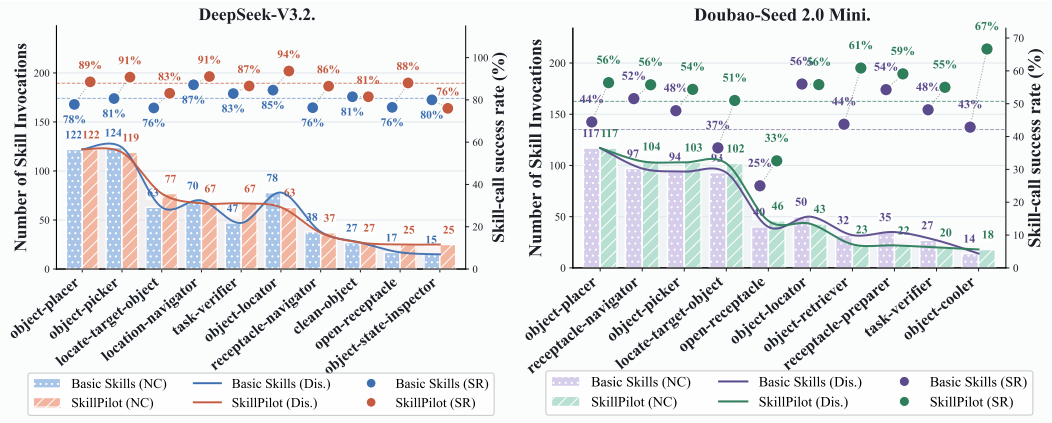
**SkillPilot also reverses the cases where the curated skill degrades the action-only baseline.** On Qwen3.5-Flash, the curated skill actually hurts success rate (39.66 vs. ReAct's 44.69%). This is because different agent models have different in-context learning behaviors and lead to different skill invocation patterns. SkillPilot recovers the drop in success rate and improves it to 46.37%, demonstrating that MCTS-based evolution can flexibly decide which way to evolve rather than just optimizing what to evolve, enabling the agent to escape from suboptimal paths.

### 3.3 Overall Performance on SkillsBench

In this subsection, we compare SkillPilot with baselines to evaluate its effectiveness on verifier-based professional tasks. The results on SkillsBench are shown in Table 2, which presents comprehensive results of human-curated or model self-generated skills, *with no trajectories from similar tasks used to support adaptation*. We only select Skill-Evolver as the skill evolution baseline, as SkillOpt requires numerous failure trajectories, which is not applicable to SkillsBench.

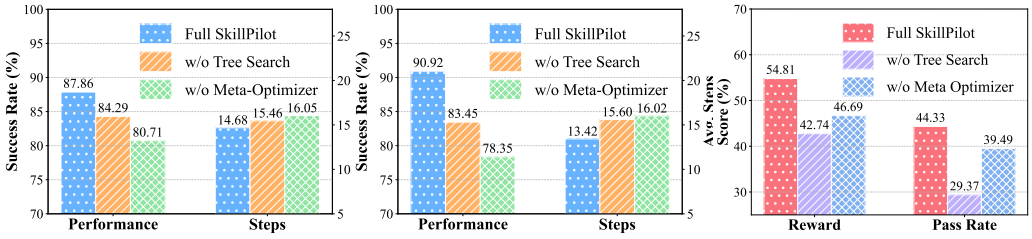
**SkillPilot significantly improves task completion on provided curated skills.** For DeepSeek-V4-Flash, SkillPilot lifts Pass Rate from Skill-Evolver's 45.75 to 54.57% and Reward from 68.59% to 74.83%, with normalized gain $g$ rising from 44.31% to 53.36%. GLM-5.3-Flash also witnesses a great relative improvement, with $g$ increasing from 21.31% to 36.29%. These results demonstrate that even without trajectories from similar tasks, SkillPilot closes the gap between the optimized skills and the target model's in-context learning behavior. We also perform a case study in Appendix E.5, showing SkillPilot effectively navigates the adaptation paths for these single hard tasks.

**SkillPilot deeply adapts the self-generated skills to the agent model itself.** Self-generated skills are noisy and rarely aligned with the task objectives initially. SkillPilot guides the adaptation to a more explorable region of the skill space, deeply improving self-generated skill execution. On GLM-5.3-Flash, the Pass Rate of the skill library is improved from 17.70% to 38.27% and $g$ from 5.81% to 29.35%, which is a significant improvement and surpasses the performance of curated skills and curated skills optimized by Skill-Evolver. This indicates that SkillPilot effectively narrows the gap between skills curated by human experts and skills generated by the base model itself.




(a) Skills Invocation by DeepSeek-V3.2. (b) Skills Invocation by Doubao-Seed 2.0 Mini.

Figure 3: Distribution & Success Rate Before/After SkillPilot. Top-10 Invoked Skills are Selected.




(a) Ablation on ALFWorld Seen. (b) Ablation on ALFWorld Unseen. (c) Ablation on SkillsBench

Figure 4: Ablation Study on ALFWorld Seen and Unseen on Success Rate and Average Steps.

### 3.4 Reusability of Optimized Skills

In this subsection, we analyze the reusability of skills optimized by SkillPilot among similar tasks. On DeepSeek-V3.2 and Doubao-Seed 2.0 Mini, we evaluate the success rate and invocation distribution of the top 10 invoked skills in ALFWorld Seen. The results are shown in Figure 3.

**SkillPilot effectively enhances the success rate of head skills, while maintaining their reusability across tasks**. For both base models, the success rate of head skills is improved substantially. The most frequently used skills "object-placer" and "object-picker" are enhanced by at least 10 percentage points on DeepSeek-V3.2. For Doubao, the most frequently used skills, including "object-placer" and "locate target object" are enhanced by 12 and 14 percentage points in success rate, respectively. These results indicate the reusability of head skills on different tasks requiring them, while adjusting the invocation of long-tail skills.

**Skills reuse is adapted to the target model by SkillPilot.** For instance, the behavior of reusing head skills is strengthened on Doubao, while the reuse of long-tail skills becomes less. On DeepSeek, the distribution of skill invocation is partially shifted. It uses "task verifier" 47 times in the evaluation. After jointly optimizing the skill library, DeepSeek is more dependent on the "task verifier" (improving by 42.55%) to coordinate with other skills to check termination. In addition, Skills with similar functions, like "locate-target-object" and "object-locator" are identified and refined.

### 3.5 Ablation Studies

In this subsection, we perform a series of ablations to analyze the contributions of main components of SkillPilot in different scenarios. For ALFWorld we select DeepSeek-V3.2 as the base model, while for SkillsBench we employ GLM-5.3-Flash. The results are shown in Figure 4.

**On ALFWorld, both the tree search and the meta-optimizer contribute to the final performance, with the meta-optimizer being more critical.** Removing tree search drops Seen/Unseen success rate from 87.86% and 90.92% to 84.29% and 83.45%, and increases steps by 0.78 and 2.18, respectively. Removing the meta-optimizer further drops the success rate by -7.15/-12.57 percentage

points. This is because the skill library of ALFWorld is large and each task invokes multiple skills, so a single edit with insufficient failure attribution may introduce semantic conflicts across skill patches. Related theoretical analysis of the meta-optimizer is shown in Appendix D.4. In contrast, the meta-optimizer prevents these conflicts and makes the tree search effective and efficient.

**On SkillsBench, both components still matter, but the MCETS is the dominant contributor.** Removing the tree search drops the Pass Rate from 44.33 to 29.37, a larger drop than removing the meta-optimizer. SkillsBench tasks involve fewer but more difficult skills per task, and each task requires a deeply explored evolution on the target model. Without tree search, only a single greedy edit is applied, leaving the upper bound of skill adaptation unexplored. SkillPilot's flexible guidance among multiple evolution paths bridges the gap between skills understanding and execution.

## 4 Related Work

**Lifecycle of Agent Skills.** Skills in the era of LLM agents share a four-stage lifecycle of representation, acquisition, retrieval, and evolution (Jiang et al., 2026b; Zhou et al., 2026a). In **skill representation**, code-based skills (ASI (Wang et al., 2025b)), natural-language skills (AWM (Wang et al., 2024), Skills-in-Context (Chen et al., 2024)) and hybrid representations (SkillNet (Liang et al., 2026)) coexist. In **acquisition**, skills can be curated by human experts like SkillsBench (Li et al., 2026b), or experience extraction such as Skill Library (Wang et al., 2026b), SkillBrew (Hu et al., 2026), EvoSkill (Alzubi et al., 2026), and ReasoningBank (Ouyang et al., 2026b), Skillx (Wang et al., 2026a) and Xskills (Jiang et al., 2026a). Skills can also be derived from memory (Zhang et al., 2026b), abstraction (Yu et al., 2026a), or generated from tasks requirement (Cai et al., 2024) and external corpus (Zhang et al., 2025), In **retrieval**, some structure-aware methods helps the agent invoke skills from large skill library, such as Skill-RL (Xia et al., 2026), GOS (Liu et al., 2026a) and AgentSkillOS (Li et al., 2026a). Generative retrieval-based methods like ToolGen (Wang et al., 2025a) internalize the skill invocation capability in the model parameters. **Skill evolution** is the last part of the skill lifecycle, enabling the skills to be self-improving from successful or failed trajectories and enhancing the agent performance, which becomes essential in the era of LLM agents.

**Skills Evolution.** Existing skill evolution methods consist of two categories. (1) discovering while optimizing. For new skills, AutoSkill (Yang et al., 2026b) introduces management strategies to improve the quality of the entire skill library, while SkillFoundry (Shen et al., 2026) discovers and evolves skills from scientific resources. In addition to training-free skill evolution workflows, some Reinforcement Learning (RL)-based methods such as SkillR1 (Vishe et al., 2026), Skill-RL (Xia et al., 2026), ARISE (Li et al., 2026c), Skill1 (Shi et al., 2026b) and SkillOS (Ouyang et al., 2026a) are developed to jointly evolve skills and the model. (2) Optimizing established skills. Existing methods mainly use recursive refinement strategies (Pan et al., 2026; Liu et al., 2026d; Yu et al., 2026b), such as EvoSkillBank (Zhou et al.), CoEvoSkills (Zhang et al., 2026c), SkillForge Liu et al. (2026c), Skillsvote (Liu et al., 2026b). The patterns of skills self-evolution also vary. For instance, PSN (Shi et al., 2026a) evolves skills in a dependency graph, Skill-Evolver (Zhang et al., 2026a) explores skills using multiple attempts, and SkillOpt (Yang et al., 2026a) optimizes them in mini-batches. SkillClaw Ma et al. (2026) and MaskClaw (Zhao et al., 2026) treat user interactions rather than working trajectories as the materials to improve skills. However, these methods mainly focus on *what* the skills should evolve, being trapped in diverse possible sub-optimal evolution paths. Uniquely, our SkillPilot breaks from single-path optimization and recasts skill evolution as meta-optimizer-driven agentic tree search, flexibly navigating evolution paths for agent models.

## 5 Conclusion

In this paper, we present SkillPilot, a meta-optimizer-guided MCETS framework for skill evolution that reformulates the evolution as agentic tree search over *where* to evolve and *which direction* to follow. A hierarchical meta-optimizer attributes failures to skill sections and code blocks and produces evidence-backed semantic gradients, guiding the direction of tree search. MCETS uses this meta-guidance as an expansion prior, flexibly explores the evolution paths, and jointly optimizes multiple skills efficiently. Across three benchmarks and multiple base models, SkillPilot consistently outperforms strong skill-evolution baselines without updating the base model, narrows the gap between skills and agent model behaviors, and maintains the optimized skills' reusability.

### AI USE STATEMENT

In this work, we used generative AI tools for [polishing words in the paper]. We have reviewed all AI-assisted work. The polished words are checked by authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### ETHICS STATEMENT

This work studies **SkillPilot**, a Monte Carlo Evolution Tree Search framework for skill adaptation in multi-turn LLM agents. Our experiments use the publicly available ALFWorld, WebShop, and SkillsBench benchmarks and do not involve new human-subject studies or the collection of personal information. The agents operate in benchmark environments.

### REPRODUCIBILITY STATEMENT

The code of this paper is available at https://anonymous.4open.science/r/SkillPilot-EF1C/. The implementation details can be found in Appendix B.4. We justify the idea of regarding the sections skill as multi-arm bandit for MCETS algorithm in Appendix D. For the theoretical support of necessity of hierarchical meta-optimizer for MCETS, including the clear assumptions, we provide a detailed demonstration in Appendix D.4 and D.5. We also provide the theoretical support of increasing weights adopted in the backpropagation stage in Appendix E.

## REFERENCES

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

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *International Conference on Learning Representations*, volume 2024, pp. 54067–54089, 2024.

Jiaao Chen, Xiaoman Pan, Dian Yu, Kaiqiang Song, Xiaoyang Wang, Dong Yu, and Jianshu Chen. Skills-in-context prompting: Unlocking compositionality in large language models, 2024. URL https://arxiv.org/abs/2308.00304.

Wei Chen, Yajun Wang, Yang Yuan, and Qinshi Wang. Combinatorial multi-armed bandit and its extension to probabilistically triggered arms. *Journal of Machine Learning Research*, 17(50): 1–33, 2016.

Pierre-Arnaud Coquelin and Rémi Munos. Bandit algorithms for tree search. *arXiv preprint cs/0703062*, 2007.

Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, et al. Cl-bench: A benchmark for context learning. *arXiv preprint arXiv:2602.03587*, 2026.

Doubao Seed. Seed 2.0, April 2026. URL https://research.doubao.com/zh/seed2.

Yi Gai, Bhaskar Krishnamachari, and Rahul Jain. Combinatorial network optimization with unknown variables: Multi-armed bandits with linear rewards and individual observations. *IEEE/ACM Transactions on Networking*, 20(5):1466–1478, 2012.

Bingzheng Gan, Yufan Zhao, Tianyi Zhang, Jing Huang, Li Yusu, Shu Xian Teo, Changwang Zhang, and Wei Shi. Master: A multi-agent system with llm specialized mcts. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 9409–9426, 2025.

Wentao Hu, Zhendong Chu, Yiming Zhang, Junda Wu, Ming Jin, Xiangyu Zhao, Yilei Shao, Yanfeng Wang, and Qingsong Wen. Skillbrew: Multi-objective curation of skill banks for llm agents. *arXiv preprint arXiv:2605.29440*, 2026.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for llm agent reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 87362–87388, 2026.

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

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. Sok: Agentic skills–beyond tool use in llm agents. *arXiv preprint arXiv:2602.20867*, 2026b.

Tor Lattimore and Csaba Szepesvári. Bandit algorithms. *Cambridge University*, 355, 2019.

Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. Organizing, orchestrating, and benchmarking agent skills at ecosystem scale. *arXiv preprint arXiv:2603.02176*, 2026a.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026b.

Yu Li, Rui Miao, Zhengling Qi, and Tian Lan. Arise: Agent reasoning with intrinsic skill evolution in hierarchical reinforcement learning. *arXiv preprint arXiv:2603.16060*, 2026c.

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, et al. Skillnet: Create, evaluate, and connect ai skills. *arXiv preprint arXiv:2603.04448*, 2026.

Huawei Lin, Peng Li, Jie Song, Fuxin Jiang, and Tieying Zhang. Muse-autoskill: Self-evolving agents via skill creation, memory, management, and evaluation. *arXiv preprint arXiv:2605.27366*, 2026.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Dawei Liu, Zongxia Li, Hongyang Du, Xiyang Wu, Shihang Gui, Yongbei Kuang, and Lichao Sun. Graph-of-skills: Dependency-aware structural retrieval for massive agent skills. *arXiv preprint arXiv:2604.05333*, 2026a.

Hongyi Liu, Haoyan Yang, Tao Jiang, Bo Tang, Feiyu Xiong, Yuyu Luo, and Zhiyu Li. Skillsvote: Lifecycle governance of agent skills from collection, recommendation to evolution. *arXiv preprint arXiv:2605.18401*, 2026b.

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

Yuxuan Liu, Zhaochen Su, Yuhao Zhang, Jiahe Guo, Zhongwei Xie, Huihao Jing, Lingyun Xie, Qing Zong, Yauwai Yim, Zhixiong Zhang, et al. Rethinking self-evolving agent skills: Feedback dynamics over multiple rounds. *arXiv preprint arXiv:2608.02636*, 2026d.

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. Skillclaw: Let skills evolve collectively with agentic evolver. *arXiv preprint arXiv:2604.08377*, 2026.

Shie Mannor and John N Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research*, 5(Jun):623–648, 2004.

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

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, et al. Skillos: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*, 2026a.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026b.

Shuai Pan, Yixiang Liu, Jiaye Gao, Te Gao, Weiwen Liu, Jianghao Lin, Zhihui Fu, Jun Wang, Weinan Zhang, and Yong Yu. Skillmas: Skill co-evolution with llm-based multi-agent system. *arXiv preprint arXiv:2605.09341*, 2026.

Xingwei Qu, Shaowen Wang, Zihao Huang, Kai Hua, Fan Yin, Rui-Jie Zhu, Jundong Zhou, Qiyang Min, Zihao Wang, Yizhi Li, Tianyu Zhang, He Xing, Zheng Zhang, Yuxuan Song, Tianyu Zheng, Zhiyuan Zeng, Chenghua Lin, Ge Zhang, and Wenhao Huang. Dynamic large concept models: Latent reasoning in an adaptive semantic space, 2026. URL https://arxiv.org/abs/2512.24617.

Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. Skillfoundry: Building self-evolving agent skill libraries from heterogeneous scientific resources. *arXiv preprint arXiv:2604.03964*, 2026.

Haochen Shi, Xingdi Yuan, and Bang Liu. Evolving programmatic skill networks. *arXiv preprint arXiv:2601.03509*, 2026a.

Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*, 2026b.

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

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Yash Vishe, Rohan Surana, Xunyi Jiang, Zihan Huang, Xintong Li, Nikki Lijing Kuang, Tong Yu, Ryan A Rossi, Jingbo Shang, Julian McAuley, et al. Skill-r1: Agent skill evolution via reinforcement learning. *arXiv preprint arXiv:2605.09359*, 2026.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, et al. Skillx: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026a.

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

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1529–1550, 2026b.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. Toolgen: Unified tool retrieval and calling via generation. In *International Conference on Learning Representations*, volume 2025, pp. 73473–73498, 2025a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. *arXiv preprint arXiv:2504.06821*, 2025b.

Yangbo Wei, Zhen Huang, Shaoqiang Lu, Junhong Qian, Qifan Wang, Chen Wu, and Lei He. Skillsmith: Co-evolving skills and tools for self-improving agent systems. *arXiv preprint arXiv:2606.01314*, 2026.

Ye Wei, Bo Peng, Ruiwen Xie, Yangtao Chen, Yu Qin, Peng Wen, Stefan Bauer, Po-Yen Tung, and Dierk Raabe. Deep active optimization for complex systems. *Nature Computational Science*, 5 (9):801–812, 2025.

Xiyang Wu, Zongxia Li, Guangyao Shi, Alexander Duffy, Tyler Marques, Matthew Lyle Olson, Tianyi Zhou, and Dinesh Manocha. Co-evolving llm decision and skill bank agents for long-horizon tasks. *arXiv preprint arXiv:2604.20987*, 2026.

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

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward, 2026. URL https://arxiv.org/abs/2602.12430.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, et al. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026a.

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

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

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

Simon Yu, Gang Li, Weiyan Shi, and Peng Qi. Polyskill: Learning generalizable skills through polymorphic abstraction for continual learning. In *International Conference on Learning Representations*, volume 2026, pp. 140298–140326, 2026a.

Zhuoyun Yu, Xin Xie, Wuguannan Yao, Chenxi Wang, Lei Liang, Xiang Qi, and Shumin Deng. Skill-ladaptor: Self-adapting skills for llm agents from trajectories. *arXiv preprint arXiv:2606.01311*, 2026b.

Z.ai. Glm-5.3, September 2026. URL https://z.ai/blog/glm-5.3.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI conference on human factors in computing systems*, pp. 1–20, 2025.

Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint arXiv:2406.07394*, 2024.

Genrui Zhang, Erle Zhu, Jinfeng Zhou, Caiyan Jia, and Hongning Wang. Skillevolver: Skill learning as a meta-skill. *arXiv preprint arXiv:2605.10500*, 2026a.

Guibin Zhang, Muxin Fu, Kun Wang, Frank Wan, Miao Yu, and Shuicheng Yan. G-memory: Tracing hierarchical memory for multi-agent systems. *Advances in Neural Information Processing Systems*, 38:12988–13018, 2026b.

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

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026d.

Wenlin Zhang, Xiangyang Li, Kuicai Dong, Yichao Wang, Pengyue Jia, Xiaopeng Li, Yingyi Zhang, Derong Xu, Zhaocheng Du, Huifeng Guo, et al. Process vs. outcome reward: Which is better for agentic rag reinforcement learning. *Advances in Neural Information Processing Systems*, 38: 58701–58729, 2026e.

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

Yanqiu Zhao, Dongying Zheng, Kaibo Huang, Yukun Wei, Zhongliang Yang, and Linna Zhou. Maskclaw: Edge-side personalized privacy arbitration for gui agents with behavior-driven skill evolution. *arXiv preprint arXiv:2605.28646*, 2026.

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, et al. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*, 2026a.

Shuaiyu Zhou, Jiaying Zhang, Taoyuan Li, Lei Shi, Fengpeng Yue, Cao Liu, and Ke Zeng. Evoskillbank: Hierarchical skill self-evolution and skill-bank governance for continual agent learning. In *Third Conference on Language Modeling*.

Yingli Zhou, Wang Shu, Yaodong Su, Wenchuan Du, Yixiang Fang, and Xuemin Lin. A comprehensive survey on agent skills: Taxonomy, techniques, and applications, 2026b. URL https://arxiv.org/abs/2605.07358.

# Technical Appendix

## Appendix Contents

**A Algorithm of SkillPilot** **16**

**B Additional Experimental Details** **16**

B.1 Introduction of Datasets . . . . . 16

B.2 Introduction to Baselines . . . . . 16

B.3 Evaluation Metircs . . . . . 18

B.4 Implementation Details . . . . . 19

B.5 Hyperparameter Analysis . . . . . 20

B.6 Prompt Templates . . . . . 21

**C Efficiency Analysis** **22**

C.1 Evolution Efficiency . . . . . 22

C.2 Storage Efficiency . . . . . 24

**D Justification of MCTS for Section-Decomposable Skill Evolution Search** **24**

D.1 Multi-Armed Bandit in Skills and MCETS for Skill Adaptation . . . . . 25

D.2 Why Meta-Optimizer Guided Tree Search? . . . . . 26

D.3 Setup: Skill Search as a Structured Bandit . . . . . 26

D.4 Linear MCETS Sample Complexity . . . . . 27

D.5 Exponential Lower Bound for Joint-Blind Methods . . . . . 29

D.6 Exponential MCETS Advantage . . . . . 30

**E Theoretical Analysis of the Back-Propagation Scheme** **30**

E.1 Setup: UCB Selection at the Root . . . . . 30

E.2 Worst-Case Deceptive Distribution . . . . . 31

E.3 Monotonicity of the Back-Propagation Weight . . . . . 31

E.4 UCB Regret Consequence . . . . . 33

E.5 Case Study: Navigating Skill Evolution via MCETS . . . . . 34

**F Limitations and Future Work** **34**

Table 3: Skill statistics across benchmarks, reported as the number of skills (#S), the number of skill files (#F), and the average number of files per skill (Avg).

| ALFWorld | | | WebShop | | | SkillsBench | | |
|---|---|---|---|---|---|---|---|---|
| Skills | Skill Files | Avg | Skills | Skill Files | Avg | Skills | Skill Files | Avg |
| 37 | 82 | 2.22 | 23 | 44 | 1.91 | 234 | 951 | 4.06 |

# A Algorithm of SkillPilot

Algorithm 1 consolidates the procedure of SkillPilot introduced in Section 2. In Phase 1, SkillPilot rolls the frozen target model $\pi_\theta$ out on the $K$ support tasks under the initial library $\mathcal{S}^{(0)}$ and retains the subset $\mathcal{F}^{(0)} \subseteq \mathcal{E}^{(0)}$ whose environment reward falls below the failure threshold $\gamma$. These trajectories, together with their invoked-skill contexts, constitute the support set on which the meta-optimizer reasons. In Phase 2, SkillPilot attaches the fix directions to the root node and initialzies the memory. SkillPilot runs for $T$ rollouts through four MCTS stages on the skill-edit tree. *Selection* descends from $v_0$ along edges ranked by the UCB rule in Eq.( 6), and the visited path is integrated into the search trajectory $\mathcal{H}_\ell \leftarrow \mathcal{H}_{\ell-1} \oplus (a_{\ell-1}, v_\ell)$ and recorded in the tree via $\mathcal{T} \leftarrow \mathcal{T} \cup \{\mathcal{H}_\ell\}$. Whenever the reached node carries residual failures $\mathcal{F}_\ell \neq \varnothing$, *refresh guidance* re-runs Algorithm 2 on the per-node memory $(\mathcal{S}_\ell, \mathcal{F}_\ell, \mathcal{M}_{v_\ell})$ so that the next expansion is conducted based on node-specific evidence. *Expansion* samples $B$ edit proposals $\Delta\mathcal{S}_{\ell,b} \sim \pi_\theta^{\text{edit}}(\cdot \mid \mathcal{G}_\ell, \mathcal{S}_\ell)$ per Eq.( 7), expand one child node $v_{\ell+1,b} = \text{Node}(\mathcal{S}_{\ell+1,b}, \ell{+}1, h_\ell \oplus \Delta\mathcal{S}_{\ell,b})$ (up to $B$ nodes), and grows the tree by $\mathcal{T}_{\ell+1} \leftarrow \mathcal{T}_\ell \cup \{\mathcal{H}_{\ell+1,b} : b \in [1, B]\}$ with $\mathcal{H}_{\ell+1,b} \leftarrow \mathcal{H}_\ell \oplus (a_{\ell,b}, v_{\ell+1,b})$. *Simulation* runs $K_{\text{sim}}$ rollouts of $\pi_\theta$ at $v_\ell$, yielding the value estimate $\widehat{V}(v_\ell) = \frac{1}{K_{\text{sim}}} \sum_{q=1}^{K_{\text{sim}}} r_q$ for nodes in Eq.( 9) and a refreshed failure set $\mathcal{F}_\ell$. *Depth-aware backpropagation* then visits every edge $(\sigma_t, a_t)$ on $\mathcal{H}_\ell$ $(t = 0, \ldots, \ell{-}1)$ and updates $Q$ with the linear-in-depth weights of Eq.( 10). In Phase 4, SkillPilot ends the loop and picks, from the final tree $\mathcal{T}_L$, the highest-reward library $\mathcal{S}^* = \mathcal{S}_{v^*}$ per Eq.( 11). At deployment, only the skill patches of $\mathcal{S}^*$ are reloaded into the agent interface, which avoids re-distributing the entire library.

# B Additional Experimental Details

## B.1 Introduction of Datasets

(1). **ALFWorld** (Shridhar et al., 2020) is an embodied household environment built on top of TextWorld, in which the agent must navigate and manipulate household objects to complete daily chores and reports both Seen and Unseen splits. (2). **WebShop** (Yao et al., 2022a) simulates a realistic online shopping site, where the agent must search a product catalog, interpret product-page text, select options, and complete a purchase under a natural-language instruction, with the standard 500 held-out shopping instructions as the test set. (3).**SkillsBench** (Li et al., 2026b) treats the skills as the unit of evaluation, with 85 tasks across eleven professional domains including healthcare, manufacturing, finance, mathematics, software engineering, and others; each task pairs a human-written instruction, and is evaluated under *Without Skills*, *Curated Skills*, or *Self-Generated Skills*. The statistics of skill liraries in the three benchmarks are listed in Table 3 The statistics show that the skill library of these benchmarks are large, within which each skill may contains multiple skill documents to be optimized.

## B.2 Introduction to Baselines

We compare SkillPilot against five representative baselines that span the spectrum from no-memory prompting to single-path skill evolution. On ALFWorld and WebShop, we compare:

**(1) ReAct** (Yao et al., 2022b) interleaves chain-of-thought reasoning with environment actions in a single trajectory and is prompted with one or two in-context examples in each task.

**(2) ExpeL** (Zhao et al., 2024) accumulates both successful and failed trajectories from training tasks into an experience pool, then distills two complementary memories from them: successful trajectories and a set of natural-language insights extracted from cross-task success–failure comparisons.

**Algorithm 1** SkillPilot: Meta-Optimizer-Guided MCTS for Skill Evolution

**Require:** Initial skill library $\mathcal{S}^{(0)}$; initial task batch $\{x_k\}_{k=1}^{K}$; failure threshold $\gamma$; search budget $T$; maximum depth $D$; branching factor $B$; simulation budget $K_{\text{sim}}$; exploration constant $c$.
**Input:** Target model $\pi_\theta$ (frozen); Initial skill library $\mathcal{S}^{(0)}$ and task prompts.
Initialize the root node $v_0$ with $\mathcal{S}_{v_0} = \mathcal{S}^{(0)}$, the root search trajectory $\mathcal{H}_0 = (v_0)$, the initial search tree $\mathcal{T}_0 = \{\mathcal{H}_0\}$, the per-node evolution memory $\mathcal{M}_{v_0} \leftarrow \varnothing$, and the rollout counter $t \leftarrow 1$.

**Phase 1: Failure-trajectory support-set collection.**
**for** $k \leftarrow 1, \dots, K$ **do**
  $\tau_k \sim \pi_\theta(\cdot \mid x_k, \mathcal{S}^{(0)})$; $r_k \leftarrow \text{Reward}(\tau_k)$
**end for**
$\mathcal{E}^{(0)} \leftarrow \{(x_k, \tau_k, r_k)\}_{k=1}^{K}$; $\mathcal{F}^{(0)} \leftarrow \{(x_k, \tau_k, r_k) \in \mathcal{E}^{(0)} : r_k < \gamma\}$
**Phase 2: Hierarchical meta-optimizer at the root.**
$\mathcal{G}_{\text{root}} \leftarrow \text{HierarchicalMetaOptimize}(\mathcal{S}_0, \mathcal{F}_0, \mathcal{M}_0)$
Attach $\mathcal{G}_{\text{root}}$ and updated memory to $v_0$.
**Phase 3: MCETS loop for skills evolution.**
**while** $t \le T$ **do**
  *(Selection)* Starting from $v_0$, descend by UCB until reaching a node $v_\ell$ at depth $\ell < D$ that is not fully expanded. At each internal state $\sigma$, pick
  $a \leftarrow \arg\max_{a \in \mathcal{A}(\sigma)} \left[ Q(\sigma, a) + c\sqrt{\frac{\ln N(\sigma)}{N(\sigma,a)}} \right]$.
  Extend the search trajectory: $\mathcal{H}_\ell \leftarrow \mathcal{H}_{\ell-1} \oplus (a_{\ell-1}, v_\ell)$, $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{\mathcal{H}_\ell\}$.
  **if** $\ell = D$ **or** $\mathcal{A}(\sigma_\ell) = \varnothing$ **then**
    Re-anchor $v_\ell$ to the deepest expandable ancestor on $\mathcal{H}_\ell$ for the remaining steps.
  **end if**
  *(Refresh guidance at $v_\ell$)* If the node has residual failures and depth permits:
  **if** $\mathcal{F}_\ell \neq \varnothing$ **and** $\ell < D$ **then**
    $\mathcal{G}_\ell \leftarrow \text{HierarchicalMetaOptimize}(\mathcal{S}_\ell, \mathcal{F}_\ell, \mathcal{M}_\ell)$
  **end if**
  *(Expansion)* At $v_\ell$, sample one edit proposal from $\mathcal{G}_\ell$ per Eq.( 7):
  $\Delta\mathcal{S}_{\ell,b} \sim \pi_\theta^{\text{edit}}(\cdot \mid \mathcal{G}_\ell, \mathcal{S}_\ell)$; $\mathcal{S}_{\ell+1,b} \leftarrow \text{Edit}(\mathcal{S}_\ell, \Delta\mathcal{S}_{\ell,b})$
  $v_{\ell+1,b} \leftarrow \text{Node}(\mathcal{S}_{\ell+1,b}, \ell{+}1, h_\ell \oplus \Delta\mathcal{S}_{\ell,b})$
  Attach $v_{\ell+1,b}$ as a child of $v_\ell$ under action $a_{\ell,b}$.
  Grow the tree: $\mathcal{H}_{\ell+1,b} \leftarrow \mathcal{H}_\ell \oplus (a_{\ell,b}, v_{\ell+1,b})$, $\quad \mathcal{T}_{\ell+1} \leftarrow \mathcal{T}_\ell \cup \{\mathcal{H}_{\ell+1,b} : b \in [1, B]\}$.
  *(Simulation)* Evaluate $v_\ell$ on $K_{\text{sim}}$ support tasks with the frozen agent $\pi_\theta$:
  **for** $q \leftarrow 1, \dots, K_{\text{sim}}$ **do**
    $\tau_q \sim \pi_\theta(\cdot \mid x_q, \mathcal{S}_\ell)$; $r_q \leftarrow \text{Reward}(\tau_q)$
  **end for**
  $\widehat{V}(v_\ell) \leftarrow \frac{1}{K_{\text{sim}}} \sum_{q=1}^{K_{\text{sim}}} r_q$; $\mathcal{E}_\ell \leftarrow \{(x_q, \tau_q, r_q)\}_{q=1}^{K_{\text{sim}}}$
  Filter the residual failures: $\mathcal{F}_\ell \leftarrow \{(x_q, \tau_q, r_q) \in \mathcal{E}_\ell : r_q < \gamma\}$.
  *(Depth-aware backpropagation)* For every edge $(\sigma_\ell, a_\ell)$ on the search path $\mathcal{H}_\ell$:
  $N(\sigma_\ell, a_\ell) \leftarrow N(\sigma_\ell, a_\ell) + 1$
  $Q(\sigma_\ell, a_\ell) \leftarrow \frac{1}{N(\sigma_\ell, a_\ell)} \sum_{v \in \mathcal{V}(\sigma_\ell, a_\ell)} w_v \, \widehat{V}(v)$, where $w_v = \text{idx}_{\mathcal{H}_v}(v)/\mathcal{L}(\mathcal{H}_v)$.
**end while**
**Phase 4: Termination and output.**
Select the best node from $\mathcal{T}_L$: $v^* \leftarrow \arg\max_{v \in \mathcal{N}_{\text{tree}}} \widehat{V}(v)$.
**return** $\mathcal{S}^* \leftarrow \mathcal{S}_{v^*}$

**Output:** Adapted skill library $\mathcal{S}^*$, together with the final search tree $\mathcal{T}_L$.

At test time ExpeL concatenates the full insight list with the retrieved similar trajectories into the prompt as in-context examples.

**(3) Skills** (Liang et al., 2026) equips the agent with a curated, section-structured skill library drawn from the SkillNet ontology, and solves tasks by retrieving and composing relevant skills. The library itself is not adapted at training time, so this baseline represents the static, agent expert-curated upper

**Algorithm 2** HierarchicalMetaOptimize: From Failure Trajectories to Patch-Level Semantic Gradients (sub-procedure of Algorithm 1)

**Require:** Candidate skill library $\mathcal{S}_v = \{S_v^{(j)}\}_{j=1}^{J}$ at search node $v$; failure-trajectory set $\mathcal{F}_v \subseteq \mathcal{E}_v$; evolution memory $\mathcal{M}_v$.
**Input:** Meta-optimizer LLM $\pi_\theta$ that shares the same model with the agent; failure threshold $\gamma$.

**Step 1: Iterative failure-to-skill attribution**.
Initialize each failure as its own cluster: $\mathcal{Z}_v^{(0)} \leftarrow \big\{(\{f\}, \mathcal{A}_f^{(0)}, u_f^{(0)})\big\}_{f\in\mathcal{F}_v}$.
**repeat**
  $\mathcal{Z}_v^{(t+1)} \leftarrow \pi_\theta^{\text{skill}}\big(\mathcal{S}_v, \mathcal{Z}_v^{(t)}\big)$
**until** $\mathcal{Z}_v^{(t+1)} = \mathcal{Z}_v^{(t)}$
Let $\{(C_m, \mathcal{A}_m, u_m)\}_{m=1}^{M}$ be the converged clusters.
**Step 2: Skill-to-patch attribution**.
**for** $m \leftarrow 1, \ldots, M$ **do**
  **for** $j \in \mathcal{A}_m$ **do**
    $\mathcal{P}_v^{(m,j)} \leftarrow \pi_\theta^{\text{patch}}(C_m, u_m, S_v^{(j)}, \mathcal{M}_v) \subseteq \{1, \ldots, n_j\}$ ▷ Attribute cluster $C_m$ to skill contents in $S_v^{(j)}$.
  **end for**
**end for**
**Step 3: Per-cluster local semantic gradient**.
**for** $m \leftarrow 1, \ldots, M$ **do**
  **for** $j \in \mathcal{A}_m$ **and** $i \in \mathcal{P}_v^{(m,j)}$ **do**
    $\mathcal{G}_v^{(j,i,m)} \leftarrow \pi_\theta^{\text{grad}}\big(s_v^{(j,i)}, (C_m, u_m, S_v^{(j)}, \mathcal{M}_v)\big)$ ▷ a concise modification direction.
  **end for**
**end for**
**Step 4: Gradient merging across clusters**.
**for** $(j, i)$ with $\mathcal{X}_v^{(j,i)} \neq \varnothing$ **do**
  $\mathcal{X}_v^{(j,i)} \leftarrow \{m : j \in \mathcal{A}_m,\ i \in \mathcal{P}_v^{(m,j)}\}$
  $\mathcal{G}_v^{(j,i)} \leftarrow \pi_\theta^{\text{merge}}\big(s_v^{(j,i)}, \{(C_m, u_m, \mathcal{G}_v^{(j,i,m)})\}_{m\in\mathcal{X}_v^{(j,i)}}\big)$ ▷ Resolve conflicting directions.
**end for**
**Step 5: Memory accumulation at node $v$.**
$\mathcal{M}_v \leftarrow \mathcal{M}_v \cup \big\{(C_m, \mathcal{G}_v^{(j,i)})\big\}_{m\in[M],\, j\in\mathcal{A}_m,\, i\in\mathcal{P}_v^{(m,j)}}$
**return** $\big\{((j,i), \mathcal{G}_v^{(j,i)}, \mathcal{M}_v) : \mathcal{X}_v^{(j,i)} \neq \varnothing\big\}$

**Output:** Meta-guidance set $\big\{((j,i), \mathcal{G}_v^{(j,i)}, \mathcal{M}_v)\big\}$; updated memory $\mathcal{M}_v$.

bound rather than a learning method, and improvements over Skills therefore correspond to gains that come purely from skill adaptation.

**(4) SkillOpt** (Yang et al., 2026a) treats the skill file as a trainable parameter outside a frozen target model and runs a forward, backward and update loop in text space: each step rolls out a minibatch of trajectories with the current skill, has a separate optimizer LLM reflect on success/failure patterns in minibatch reflection.

**(5) Skill-Evolver** (Zhang et al., 2026a) is an online skill learning loop driven by a single meta-skill that governs authoring, deployment, and refinement. Like SkillOpt, Skill-Evolver commits to a single edit chain per round and cannot compose alternative patches along different paths.

On SkillsBench, we compare **(1) Without Skills (ReAct)**, **(2) With Skills** (provided by Skillsbech), and **(3) Skill-Evolver** against our SkillPilot.

### B.3 Evaluation Metircs

In this subsection, we report the evaluation metrics across the three benchmarks.

**(1) ALFWorld.** Each task $x$ yields a binary terminal outcome $\mathrm{succ}_x \in \{0,1\}$ returned by the environment after the agent's trajectory terminates (or aborts with an error). The *success rate* and the *average number of steps* over a task family $\mathcal{X}$ of size $N$ are

$$\mathrm{SR}_{\mathrm{ALF}} = \frac{1}{N}\sum_{x\in\mathcal{X}} \mathrm{succ}_x, \qquad \overline{L}_{\mathrm{ALF}} = \frac{1}{N}\sum_{x\in\mathcal{X}} \big|\tau(x)\big|,$$

where $\big|\tau(x)\big|$ denotes the number of grounded actions in the trajectory produced by $\pi_\theta$ on $x$.

**(2) WebShop.** After the agent issues `click[Buy Now]` on task $x$, WebShop emits a deterministic scalar reward $r_x \in [0,1]$. Following (Yao et al., 2022a), let $U_{\mathrm{att}}, U_{\mathrm{opt}}$ be the sets of attributes/options mentioned in the instruction, $Y_{\mathrm{att}}, Y_{\mathrm{opt}}$ be the corresponding sets satisfied by the purchased product $y$, and $r_{\mathrm{type}} \in [0,1]$ be the type-reward from a TextMatch heuristic that aligns coarse-grained category, fine-grained category, and product title. The reward combines type match, attribute match, option match, and a price budget indicator as

$$r_x = r_{\mathrm{type}} \cdot \frac{\big|U_{\mathrm{att}} \cap Y_{\mathrm{att}}\big| + \big|U_{\mathrm{opt}} \cap Y_{\mathrm{opt}}\big| + \mathbf{1}[y_{\mathrm{price}} \le u_{\mathrm{price}}]}{|U_{\mathrm{att}}| + |U_{\mathrm{opt}}| + 1}. \tag{12}$$

Then We report three task-level evaluation metrics:

$$\mathrm{TCR}_{\mathrm{WS}} = \frac{1}{N}\sum_{x\in\mathcal{X}} \mathbf{1}\big[r_x > 0\big], \quad \overline{r}_{\mathrm{WS}} = \frac{1}{N}\sum_{x\in\mathcal{X}} r_x, \quad \overline{L}_{\mathrm{WS}} = \frac{1}{N}\sum_{x\in\mathcal{X}} \big|\tau(x)\big|,$$

where $\mathbf{1}(\cdot)$ is the indicator function, $\gamma_{\mathrm{WS}} = 0$ is the WebShop success threshold supplied by the benchmark and $\overline{r}_{\mathrm{WS}}$ is the average reward calculated by the benchmark. $\overline{L}_{\mathrm{WS}}$ is the average execution steps that measures efficiency.

**(3) SkillsBench.** The benchmark does not ship a scalar reward per trajectory; each task $i \in \mathcal{I}$ is graded by a deterministic predefined verifier whose output is a finite set of assertions. We follow the partial-credit assertion-averaging formulation: let $\mathcal{V}_i = \{e_{i,1}, \ldots, e_{i,N_i}\}$ denote the assertion suite shipped with task $i$ and let $y_i^{(t)}$ be evolved skills produced by the agent on trial $t$. The per-trial reward on task $i$ is the fraction of assertions satisfied,

$$\widetilde{R}(y_i^{(t)}, \mathcal{V}_i) \triangleq \frac{1}{|\mathcal{V}_i|}\sum_{k=1}^{|\mathcal{V}_i|} \mathbf{1}[e_{i,k}(y_i^{(t)})], \quad \widetilde{R} \in [0,1]. \tag{13}$$

The trial is counted as a success whenever $\widetilde{R} = 1$, i.e. when all $|\mathcal{V}_i|$ assertions are simultaneously satisfied. Aggregating over the $A$ independent trials per task, the mean reward and pass rate are

$$\overline{r}_i = \frac{1}{A}\sum_{t=1}^{A} \widetilde{R}(y_i^{(t)}, \mathcal{V}_i), \qquad \mathrm{PR}_i = \frac{1}{A}\sum_{t=1}^{A} \mathbf{1}\big[\widetilde{R}(y_i^{(t)}, \mathcal{V}_i) = 1\big], \tag{14}$$

with $A = 5$ under Curated Skills and $A = 3$ under Self-Generated Skills, which is proven to stably reflect the performance of skills Li et al. (2026b). Reporting across the $|\mathcal{I}| = 85$ benchmark tasks yields the three reported quantities

$$R = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} \overline{r}_i, \quad \mathrm{PR} = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} \mathrm{PR}_i, \quad g = \frac{\mathrm{PR}_{\mathrm{skill}} - \mathrm{PR}_{\mathrm{vanilla}}}{1 - \mathrm{PR}_{\mathrm{vanilla}}} \tag{15}$$

where $R$ is the average reward of all tasks, while PR is the average pass rate of all tasks, and $g$ is the normalized gain of using skills. $\mathrm{PR}_{\mathrm{vanilla}}$ denotes the pass rate when the agent employs the ReAct pattern and does not use skills, and $\mathrm{PR}_{\mathrm{skill}}$ is the pass rate when the agent use skills.

### B.4 Implementation Details

**(1) ALFWorld and Webshop.** We run MCETS for 20 iterations per task with $B = 2$ children per expansion. For ALFWorld and WebShop, we sampled 200 tasks from the training set, and among these 200 tasks, we randomly collected $K = 20$ failure trajectories for the simulation at the root node in MCETS. The exploration constant $c$ in UCB1 is set to 0.7, and the maximum depth tree search is $D = 10$. The failure threshold adopted to collect failed trajectories from the support set

Table 4: Hyperparameter Analysis on ALFWorld. We consider the budget of the support-trajectory $N$, the budget of MCETS iteration $T$, and the maximum branching factor $B$. Rows are grouped by the width of the tree in MCETS, i.e., per-task explored evolution branches. "$\Delta^{\%}_{rel}$" reports the relative improvement of average performance in each row compared with the first row.

| Configuration | ALFWorld Seen | | ALFWorld Unseen | | |
|---|---|---|---|---|---|
| | Reward | Avg. Steps | Reward | Avg. Steps | Avg./$\Delta^{\%}_{rel}$ |
| ***Number of Explored Evolution Nodes* $\leq$ *10*** | | | | | |
| $N=5, T=5, B=2$ | 65.71 | 18.15 | 70.90 | 18.50 | 68.31 $\Delta_{\text{base}}$ |
| $N=10, T=5, B=2$ | 68.57 | 16.66 | 71.64 | 17.16 | $70.11_{+2.64\%}$ |
| $N=10, T=10, B=2$ | 73.57 | 17.06 | 69.40 | 17.76 | $71.49_{+4.66\%}$ |
| ***Number of Explored Evolution Nodes* = *20*** | | | | | |
| $N=20, T=20, B=3$ | 72.14 | 17.26 | 75.37 | 16.97 | $73.76_{+7.39\%}$ |
| $N=20, T=20, B=2$ | 70.00 | 17.26 | 76.12 | 16.19 | $73.06_{+6.95\%}$ |
| $N=40, T=20, B=2$ | 73.57 | 16.71 | 74.43 | 17.53 | $74.00_{\mathbf{+8.33\%}}$ |
| ***Number of Explored Evolution Nodes* = *40*** | | | | | |
| $N=20, T=40, B=3$ | 72.14 | 17.26 | 75.37 | 16.97 | $73.76_{+7.39\%}$ |
| $N=20, T=40, B=2$ | 70.00 | 17.26 | 76.12 | 16.19 | $73.06_{+6.95\%}$ |
| $N=40, T=40, B=3$ | 70.71 | 16.83 | 76.87 | 17.03 | $73.79_{+8.02\%}$ |

for ALFWorld is set to $\gamma = 1$, while for Webshop we set $\gamma = 0.2$. For the evaluation process, both benchmarks adopt 30 as the maximum number of steps for each task's evaluation, and we evaluate all methods of skill evolution conditioned on DeepSeek V3.2 (Liu et al., 2025), Doubao-Seed 2.0 mini (Doubao Seed, 2026), and Qwen3.5-Flash (Team, 2026). For all skill-based baselines, we adopt the skill library curated from SkillNet (Liang et al., 2026). The maximum number of skills invoked for each task is set to 6. Since the initial skills of Alfworld and WebShop are text-based skills, the section partition for the edit unit is decided by "#" in markdown files. **(2) SkillsBench.** We run MCETS for 10 iterations per task with $B = 2$ children per expansion. Additionally, for curated skills, the results are averaged over 5 trials, while for self-generated skills by the base model, the results are averaged over 3 trials. The exploration constant $c = 0.7$ and the failure threshold is set to $\gamma = 1$. At the root node, the failure trajectories are collected from these trials and are utilized in the simulation process. For SkillsBench, we use DeepSeek-v4-flash (Xu et al., 2026) and GLM-5.3-flash (Z.ai, 2026). In SkillsBench, the section partition for the edit unit is also decided by "#" in markdown files, and the content and range of each code block is decided by the base LLM model. ach edited patch is localized by its line indices. Consistently, all inference of skill evolution and evaluation stages are performed by the same base models. The batch size of iterative merging in hierarchical meta-optimizer is set to 3. The evolution depths of methods based on a single evolution path are controlled the same way as in SkillPilot, and the maximum number of patches $m_a$ at each round is 3 for all the evolution methods.

For **SkillOpt** (Yang et al., 2026a), on both ALFWorld and WebShop we set the reflection minibatch size to 5, the merge batch size to 5, which is consistent with the setting of mini-batch-based failure clustering. We set the textual learning rate to 3 edits per step, which is also consistent with SkillPilot; we depart from the default two-model (frozen target and external optimizer teacher) setup and let the agent model reflect on its own trajectories. For **Skill-Evolver** (Zhang et al., 2026a), on SkillsBench we run $R$=1 refinement round with 3 strategy-diversified exploration trials and 5 validation trials per task for curated skills and 3 trials for self-generated skills, yielding 6 and 4 Harbor trials per task in total, respectively; on ALFWorld and WebShop we keep 3 exploration trials and refinement round $R$=1 but expand the validation budget to 20 in each task, which is consistent with the setting of SkillPilot.

### B.5 Hyperparameter Analysis

In this subsection, we analyze the effects of hyperparameters of tree search from the aspects of the budget of the support-trajectory $N$, the budget of MCETS iteration $T$, and the maximum branching factor $B$. The results are shown in Table 4.

Increasing the support-trajectory count $N$, the iteration budget $T$, and the tree width $B$ can each improve SkillPilot to some extent. For example, raising $N$ from 5 to 40 lifts the average reward from 68.31 to 74.00, as more trajectories provide useful experience for skills evolution. In addition, growing $T$ from 5 to 40, together with $N$, widens the search tree, and brings gains to the skills performance. However, these improvements do not grow without bound: once $N$ and $T$ reach 40, the skill-evolution ceiling is almost reached, and additional trajectories or exploration iterations bring little or no further benefit. For instance, at $N = 20$ with $B = 3$, the overall performance does not improve as $T$ is increased from 20 to 40 (73.76 in both cases). Likewise, with the same exploration budget, widening the tree to $B = 3$ yields little additional gain unless the support set is also large enough. Only with a larger trajectory set does choosing a wider tree truly benefit SkillPilot's exploration.

### B.6 Prompt Templates

We provide the prompt templates for failure clustering and the meta-optimizer for hierarchical attribution, and templates of the edit actions for MCTS expansion. All prompts are wrapped in `[SYSTEM]` / `[USER]` blocks. The prompts for task execution in the evaluation stage are consistent with previous work (Liang et al., 2026).

**Prompt for Failure Clustering.** This prompt implements Step 1 of Algorithm 2. The injected {`groups`} carries the previous clustering state $\mathcal{Z}^{(t)}$ so that each call returns an incremental refinement $\mathcal{Z}^{(t+1)}$. The rigid JSON schema guarantees a deterministically parseable output that skill-to-patch attribution can consume without free-text interpretation.

**Prompt for Failure Clustering**

```
You are analyzing skill optimization failures.
The following cases did not complete the task.
You should group them and summarize the **root cause that led
to the task failure**, and suggest possible fix directions to
avoid the failure.
For example, analyze reason why the tasks fail even with max
steps...
Merge/refine the grouped Failure analysis and map each category
 to likely problematic skill areas.
Existing groups:
{groups}
Input record JSON:
{json.dumps(prompt_payload, ensure_ascii=False)}
Return JSON only with schema:
{{
  "groups": [
    {{
      "category": "short_id",
      "count": 0,
      "case_indices": [1,2],
      "suspected_skill_areas": ["skill-a", "skill-b"],
      "root_cause": "...",
      "impact": "...",
      "fix_directions": ["...", "..."]
    }}
  ]
}}
```

Figure 5: Prompt Template for Merging Failures into Clusters.

**Prompt for Meta-Optimizer.** This prompt implements Step 3 of Algorithm 2, which generates per-cluster local semantic gradients. The three inputs jointly constrain the edit space: the failure cluster report ties the suggestion to the current node's failure diagnosis, the modification history prevents redundant rewrites of the same skill contents, and the candidate editable sections restrict attention to the patches already attributed by Step 2. The action vocabulary is restricted to

{add, remove, modify, create} with at most three edits per call, enforcing the minimal and generalizable principle that keeps MCETS expansion sparse. Since the reward for the WebShop dataset is continuous rather than binary, the prompt for WebShop shown in Figure 7 additionally injects the four official reward components (attribute, option, price, type match) so the LLM knows which dimensions the surrogate reward decomposes along.

**Prompt for Meta-Optimizer in ALFWorld and SkillSBench**

```
You are a skill editor for ALFWorld.
Inputs:
1) report.md content:
{node_report_text.strip()}
2) current path modification history (JSON string):
{modification_record_for_prompt}
3) candidate editable sections (JSON list):
{json.dumps(section_records, ensure_ascii=False)}

Task:
- Propose up to 3 file/section edits for this expansion step.
- Use only actions: add, remove, modify, create.
- Keep edits concise and directly tied to the given error cases
.

Return JSON only:
{{
    "modification_suggestions": [
        {{
            "target_file": "
relative_file_path_or_new_skill_name",
            "target_section": "section_id_or_heading",
            "action": "add|remove|modify",
            "content": "concise edit instruction"
        }}
    ]
}}
```

Figure 6: Prompt Template for Meta-Optimizer Proposing Fix Directions in ALFWorld and SkillsBench.

**Prompt for Edit Actions.** This prompt executes the edit actions at the end of the expansion stage in Algorithm 1. Unlike the two previous prompts that produce suggestions, this prompt asks for the full revised section content, so the MCETS expansion stage can apply the edit atomically without further parsing. The instructions "keep valid syntax / preserve useful original content" guard against accidental deletion of working parts of the skill.

# C Efficiency Analysis

## C.1 Evolution Efficiency

In this subsection, we evaluate the evolution efficiency in terms of token consumption across the three benchmarks. For ALFWorld and Webshop, we measure the total token cost incurred until the allocated token budget is exhausted. For SkillsBench, we analyze the cost differences using six representative tasks that were successfully evolved. The token consumption across the three baseline models is illustrated in Figure 9. As demonstrated, SkillPilot reduces token overhead by jointly optimizing multiple skills within a shared context. This joint optimization minimizes the input tokens required compared to optimizing individual skill documents (as in SkillOpt) or skill artifacts (as in Skill-Evolver). Ultimately, these results underscore the necessity of joint skill optimization for large-scale skill libraries, achieved by formulating sections and code blocks as bandits within the MCETS framework.

**Prompt for Meta-Optimizer in Webshop**

```
You are a skill editor for WebShop.
Inputs:
1) report.md content:
{node_report_text.strip()}
2) current path modification history (JSON string):
{modification_record_for_prompt}
3) candidate editable sections (JSON list):
{json.dumps(section_records, ensure_ascii=False)}
Task:
- Propose up to 3 file/section edits for this expansion step.
- Use only actions: add, remove, modify, create.
- Keep edits concise and directly tied to the given bad cases
so that the task completion and success rate can be improved.
- IMPORTANT: The edits should be **minimal** and generalizable
(Do not modify too many texts). The edit can solve the problem
and can preserve the successful trajectories.
The following aspects should also be considered to improve the
quality of task execution.
- Attribute Match: Measures the overlap between the product's
actual attributes and the specific attributes requested in the
instruction.
- Option Match: The selected product options (e.g., color, size
, flavor) should strictly align with the user's specifications.
- Price Constraint: The product's price should be within the
specified budget limit.
- Type Match: Ensures core product identity alignment (e.g.,
distinguishing "butter" from "plant-based meat" despite shared
attributes like "non-GMO").

Return JSON only:
{{
    "modification_suggestions": [
        {{
            "target_file": "
relative_file_path_or_new_skill_name",
            "target_section": "section_id_or_heading",
            "action": "add|remove|modify",
            "content": "concise edit instruction"
        }}
    ]
}}
```

Figure 7: Prompt Template for Meta-Optimizer Proposing Fix Directions in WebShop.

In addition to the perspective of token cost, we also evaluates the evolution efficiency on the number of evolution iterations. Under the setting where the optimizer sees only the failed trajectory set $\mathcal{F}_v$ produced by the previous round, the single-path reflective evolution method, Skill-Evolver may rewrite the same failed skill contents along one trajectory, so the agent can accumulate and oscillate around an existing error: it fails to locate the skill contents that needs to change, or to align the new text with the target model's own in-context learning behavior, and so it has to spend extra iterations retrying the same neighborhood. Tree-structured exploration SkillPilot flexibly selects the evolution path by dynamic value assignment: whenever one branch underperforms, the MCETS back-propagation immediately downweights it and the next round budgets its rollouts on a more promising sibling. We select six representative SkillsBench tasks on which both of the methods successfully evolve, and report the costs of both methods that reach the *same* final reward. As shown in Table 5, SkillPilot cuts the cumulative evolution cost substantially: the average token cost drops from $6.95\,\mathrm{M}$ to $3.71\,\mathrm{M}$). The substantial improvement in efficiency is mainly attributed to two reasons: (1) The number of iterations in SkillPilot is lower, as it can effectively exploit high-quality skill patches and escape from the evolution paths with low values. With fewer iteration rounds

**Prompt for Edit Actions**

```
You are editing one WebShop skill subsection.

Target file: {target_file}
Target section id: {target_section.get("id", "")}
Target section heading: {target_section.get("heading", "")}
Target section content to be modified: {section_text.strip()}
Operation: {action}
Operation guideline: {operation_hint}
Instruction: {content}

Return ONLY the full revised section content for this section.
Keep valid syntax/format and preserve useful original content.
If this section has a markdown heading line, keep it.
```

Figure 8: Prompt Template for Edit Actions.

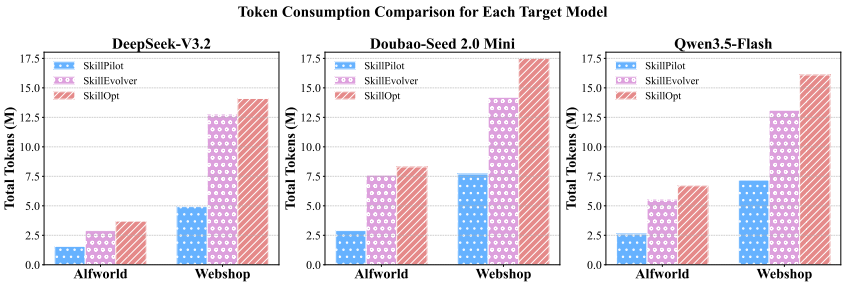



Figure 9: Token cost comparison for three target base models on ALFWorld and Webshop

and a smaller depth, a high-quality and highly adaptable skill document can be evolved. (2) Using the same number of iterations, SkillPilot can produce more high-quality skills, saving tokens for trajectory rollouts.

### C.2 Storage Efficiency

The MCETS expansion in Algorithm 1 only apply the small subset of patches generated by the hierarchical meta-optimizer; the remaining documents are reused unchanged from the curated library. In SkillPilot, persistent storage scales with the number of patches rather than with the total library size. Table 6 reports the on-disk footprint of the per-model adapted library across ALFWorld and WebShop, where "Total Savings" is the fraction of the library tokens *not* shipped at deployment time, i.e. $\big(1 - (\text{ALF Tokens} + \text{WS Tokens})/\text{Lib. Tokens}\big) \times 100\%$.

According to the results, the full revised library totals 221–494 KB (52k–115k tokens), but the deployed patch payload is only 14–20 KB on ALFWorld (3k–5k tokens, less than $10\%$ of the library tokens) and 32–33 KB on WebShop (7k tokens, less than $15\%$ of the library tokens). At deployment the agent loads only the patches together with their $(j, i)$ location index, reducing the persistent footprint by more than an order of magnitude compared with reflective optimizers that rewrite the entire skill library at every adaptation round, while still allowing joint, deep adaptation across multiple skills.

## D Justification of MCTS for Section-Decomposable Skill Evolution Search

This appendix introduces the innovative idea of MCTS for skills evolution and gives a self-contained theoretical justification for why MCTS is the right tool for skill optimization, complementing the back-propagation analysis of Section D.2. The argument is a standard combinatorial multi-armed

Table 5: Task-level evolution cost for all attempts to reach the same final reward. "Iter." counts the number of optimization rounds; "Tokens" reports the total of input output tokens of the evolution loop, in millions (M). The best results are bold.

| | | Skill-Evolver | | SkillPilot | |
|---|---|---|---|---|---|
| **Task** | **Reward** | **Iter.** | **Tokens (M)** | **Iter.** | **Tokens (M)** |
| court-form-filling | 1.0 | **7** | 3.58 | 8 | **2.86** |
| azure-bgp-oscillation-route-leak | 1.0 | 10 | 1.63 | **8** | **1.24** |
| dapt-intrusion-detection | 0.8 | 10 | 3.96 | **8** | **2.28** |
| enterprise-information-search | 0.73 | 13 | 25.95 | **10** | **10.07** |
| earthquake-phase-association | 0.4 | 12 | 1.66 | **8** | **1.44** |
| lab-unit-harmonization | 1.0 | 12 | 4.93 | **10** | **4.36** |
| **Average Cost** | - | 10.67 | 6.95 | **8.83** | **3.71** |

Table 6: On-disk footprint of the per-model adapted skill library on ALFWorld and WebShop. "Size (KB)" and "Tokens" of the Full Skill Library are the totals over every skill document shipped with the library including ALFWorld and Webshop. SkillPilot only stores the skill patches together with their $(j, i)$ location index, separately for the two environments.

| | Full Skill Library | | ALFWorld patches | | WebShop patches | | |
|---|---|---|---|---|---|---|---|
| **Target Model** | **Size (KB)** | **Tokens** | **Size (KB)** | **Tokens** | **Size (KB)** | **Tokens** | **Total Savings** |
| Doubao | 274.34 | 63,584 | 14.09 | **2,962** | 32.88 | **7,080** | **84.21%** |
| DeepSeek | 232.57 | 54,991 | 19.24 | **4,189** | 32.67 | **7,441** | **78.85%** |
| Qwen3.5-Flash | 221.10 | 52,837 | 19.95 | **4,565** | 33.02 | **7,374** | **77.41%** |

bandit (CMAB) reduction (Zhang et al., 2024; Chen et al., 2016; Gai et al., 2012; Mannor & Tsitsiklis, 2004; Lattimore & Szepesvári, 2019; Coquelin & Munos, 2007) and keeps the proofs compact.

### D.1 Multi-Armed Bandit in Skills and MCETS for Skill Adaptation

. In this subsection, we introduce the idea of SkillPilot in detail, which starts from a meta-learning task for adapting the skill library to a target model, to a meta-optimizer-guided tree search navigating the evolution path, and finally to the idea of sections/code blocks as a multi-armed bandit for the tree search. These parts of the idea form a closed loop for the skill library adaptation, navigating the search path from efficient exploration by pulling the arms of combinatorial bandits, i.e., edit actions on multiple skills. The idea of SkillPilot is shown in Figure 10.

**Meta Learning Task for Skill Library Adaptation.** As shown in Figure 10, the skill library is regarded as a trainable representation, and should be optimized in different directions for different target models using a small support task. With the guidance of a hierarchical meta-optimizer, SkillPilot applies patches to skills and adapts the skills to the target model. However, since the skill library contains multiple skill documents, optimizing it with few-shot trajectories may lead to accumulated errors or conflicts in existing methods based on a single skill evolution path. We therefore consider using the meta-optimizer-guided MCETS.

**Meta-Optimizer-Guided MCETS for Meta Learning Task.** We employ MCETS to optimize the skill library based on the support task, which exhibits high sample efficiency. Each action of editing the skill library with skill patches is a step of optimization in the meta-learning task. Within a small support set, SkillPilot fully explores and flexibly selects the path with the skill patches adapted to the target model, preventing the adaptation from being trapped in suboptimal paths. The suboptimal evolution is marked as pink arrows in Figure 10, leading to a higher error rate. For MCETS, the actions (arms) and tree nodes (bandits) should also be defined to guarantee the efficient adaptation of SkillPilot.

**Skill Contents as Multi-Armed Bandits for MCETS.** Directly treating the whole skill library as a bandit may lead to exponential explosion in samples and high token costs. Inspired by the highly structured nature of skill documentation, we consider each section or code block of a skill as a multi-armed bandit (MAB), where each arm represents an edit to the section or code block. In

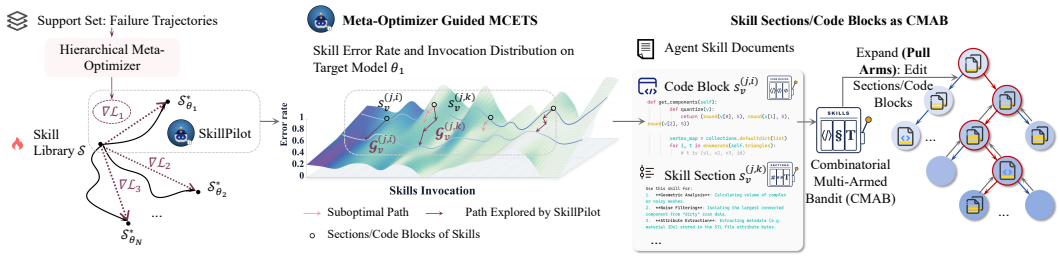



Figure 10: The detailed mechanism of SkillPilot. (1) The meta learning task: SkillPilot aims to adapt the skill library to the target model based on the support set. (2) Meta-Optimizer Guided MCETS: To solve the meta-learning task, SkillPilot optimizes the skill library with MCETS, flexibly selecting adaptation path of higher value. (3) Skill contents as MAB: by pulling the arm (apply patches) on sections/code blocks (bandits), SkillPilot efficiently explores patches across multiple skills, which is also tailored to MCTS.

MCETS, each tree node (also a skill library state) is a CMAB instance; The edge of the tree is a super arm, representing the joint editing action, which is also the edit actions applying patches during the expansion phase of nodes; Each section within a node is an arm group of the CMAB, which is a local small bandit. With this mechanism, SkillPilot can navigate the evolution path efficiently and effectively. Detailed theoretical analysis can be found in the following sections.

### D.2 Why Meta-Optimizer Guided Tree Search?

The meta-optimizer and Monte Carlo Evolution Tree Search address complementary sources of search difficulty. The meta-optimizer decouples the failures and maps them to a small set of grounded skill contents (such as sections and code blocks) and semantic edit directions. MCETS then searches for edit actions and explores a high-quality evolution path, which is necessary when edit utility is path-dependent or when promising patches conflict after composition. Execution reward serves as the feedback for SkillPilot to flexibly select the evolution path for LLM agents. The algorithm works due to ***the complex but regular structure of the skill library, where sections and code blocks can be regarded as multi-armed bandits and actions that can be searched efficiently by MCETS***. To demonstrate this, we perform a detailed theoretical analysis. Under a section-decomposable reward model, Appendix D formalizes the sample-efficiency advantage of structured search over methods that treat the entire library as one mutation unit. Appendix E separately gives the assumptions and worst-case analysis for the linear depth weight in Equation 10. These results motivate the design; the algorithm itself requires neither reward decomposition nor access to gradients.

### D.3 Setup: Skill Search as a Structured Bandit

A skill is a tuple $\mathcal{S} = (e_1, \ldots, e_N)$ of section-level components, with section set $\mathcal{E}_i$ of size $b_i$ for section or code block $i$. The joint search space is

$$\mathcal{S} = \prod_{i=1}^{N} \mathcal{E}_i, \qquad |\mathcal{S}| = \prod_{i=1}^{N} b_i. \tag{16}$$

The reward of a skill on the adaptation task is the agent's success rate $R(\mathcal{S}) \in [0, 1]$. Given a rollout budget $B$, the goal is to identify an $\epsilon$-optimal skill $\mathcal{S}^\star = \arg\max_{\mathcal{S}\in\mathcal{S}} R(\mathcal{S})$ with $R(\mathcal{S}^\star) - R(\mathcal{S}) \le \epsilon$ using as few rollouts as possible. Since the semantic space is continuous (Qu et al., 2026), the corresponding discrete search space of skill edit actions is extremely vast, preventing existing methods from evolving high-quality skills adapted to the target model.

**CMAB interpretation.** Treating the skill as a hyper-arm and each section as a base arm, the problem is a combinatorial bandit with arm set $\mathcal{S}$ and combinatorial reward $R$ (Chen et al., 2016; Lattimore & Szepesvári, 2019). Two structural assumptions distinguish MCETS from generic CMAB algorithms and underwrite its efficiency.

**Assumption 1** (Weakly additive reward)**.** There exist per-section reward functions $r_i \colon \mathcal{E}_i \to [0,1]$ and a global coupling term $\epsilon_{\max} \geq 0$ such that, for every $\mathcal{S} = (e_1, \ldots, e_N) \in \mathcal{S}$,

$$\left| R(\mathcal{S}) - \sum_{i=1}^{N} r_i(e_i) \right| \;\leq\; \epsilon_{\max}. \tag{17}$$

When $\epsilon_{\max} = 0$ the reward is exactly additive across sections; for $\epsilon_{\max} > 0$ the section structure is *approximate*, and $\epsilon_{\max}$ is the slack that absorbs cross-section interaction.

**Assumption 2** (Non-degenerate per-section reward range)**.** For every section $i$, the reward range satisfies

$$w_i \;\triangleq\; \max_{e \in \mathcal{E}_i} r_i(e) - \min_{e \in \mathcal{E}_i} r_i(e) \;\geq\; w \;>\; 0$$

for an absolute constant $w$. The constant $w$ lower-bounds the per-section signal that the bandit needs in order to identify the optimal skill; it rules out degenerate sections whose edits are interchangeable.

Assumption 1 is empirically related to practical cases in skill libraries: sections of a modern skill, including workflow, tools, examples, and description, are designed to cooperate with each other and independently contribute to the total reward. In addition, the meta-optimizer attributes failures at section granularity, so cross-section interactions are local and sparse. The coupling budget $\epsilon_{\max}$ captures the residual dependence (e.g. a tool entry that only makes sense given a particular workflow step) and remains a free parameter of the theory. Assumption 2 is a standard non-degeneracy condition for bandit identification (Auer et al., 2002; Lattimore & Szepesvári, 2019).

### D.4 Linear MCETS Sample Complexity

**Theorem 1** (MCETS achieves $\tilde{O}(N)$-sample complexity for skill evolution search)**.** *Under Assumptions A1 and A2, the section-level MCETS algorithm that runs UCB1 (Auer et al., 2002) independently at every depth $i$ (i.e. inside the subtree rooted at each section) identifies an $(\epsilon + \epsilon_{\max})$-optimal skill with probability at least $1 - \delta$ after*

$$N_{\mathrm{MCETS}}(\epsilon, \delta) \;=\; O\left( \frac{\sum_{i=1}^{N} b_i \, \log(b_i N/\delta)}{w^2} \right) \tag{18}$$

*rollouts.*

*Proof.* We reduce section-level MCETS to $N$ independent $b_i$-armed best-arm identification problems and apply the UCB1 analysis of (Auer et al., 2002; Lattimore & Szepesvári, 2019).

*Step 1 (Section-level bandit).* Fix a section $i$. The MCTS-based approach maintains a UCB1 and selects edit $e \in \mathcal{E}_i$ to maximize the index $Q_i(e) + c\sqrt{\ln T / N_i(e)}$, where $Q_i(e)$ is the empirical mean reward of rollouts that selected edit $e$ at section $i$ and $N_i(e)$ is its visit count. Under Assumption 1 the joint reward decomposes as $R(\mathcal{S}) = \sum_j r_j(e_j) + \eta(\mathcal{S})$ with $|\eta| \leq \epsilon_{\max}$. All rollouts that reach depth $i$ share the same distribution over earlier sections, so conditioning on $e_i = e$ gives

$$\mathbb{E}[R \mid e_i = e] \;=\; r_i(e) + C_i + \xi, \qquad |\xi| \leq \epsilon_{\max},$$

where $C_i = \mathbb{E}\big[\sum_{j<i} r_j(e_j)\big]$ is the expected contribution of sections $1, \ldots, i-1$ under the rollout policy and does not depend on $e$. UCB1 therefore sees an effective per-section reward gap of at least $w$ on $r_i(\cdot)$, according to Assumption 2.

*Step 2 (Section-level sample complexity).* Apply Auer et al. (2002) Theorem 1 to the $b_i$-armed bandit at section $i$: its expected regret after $n$ plays satisfies

$$\mathbb{E}\big[\mathcal{R}_i(n)\big] \;\leq\; \frac{8 \ln n}{\Delta_i} \;+\; \left(1 + \tfrac{\pi^2}{3}\right) \sum_{j=1}^{b_i} \Delta_j, \tag{19}$$

where $\Delta_i \overset{\text{def}}{=} \mu^* - \mu_i$ and $\mu_i$ is the reward expectation for skill section $i$ and $\mu^*$ is any optimal arm in the bandit. Because $\mathcal{R}_i(n) = \sum_{i<i^\star} \Delta_j T_j(n) \geq \Delta_i T_i(n)$ (non-negativity of the other terms)

where $i^\star$ is the optimal arm, i.e., the optimal edit strategy on the section $i$, substituting equation 19 and dividing by $\Delta_i$ yields

$$\Delta_i \, \mathbb{E}\big[T_i(n)\big] \;\le\; \mathbb{E}\big[\mathcal{R}_i(n)\big] \;\le\; \frac{8\ln n}{\Delta_i} \;+\; \sum_{j\neq i} \frac{8\ln n}{\Delta_j} \;+\; O(b_i). \tag{20}$$

Hence in expectation each *suboptimal* arm $i$ is pulled at most $\mathbb{E}[T_i(n)] = O\big(b_i \ln n/\Delta_i^2\big)$ times. Then for any $\delta' \in (0,1)$, Markov's inequality then turns equation 20 into:

$$\Pr\left[T_i(n) \;\ge\; \frac{b_i \ln n}{\Delta_i^2 \delta'}\right] \;\le\; \delta', \tag{21}$$

where $K_1$ is a constant. The probability of this event is also equivalent to:

$$\Pr\left[\forall j \neq i^* : T_j(n) \le O\left(\frac{b_i \ln n}{\Delta_j^2 \delta'}\right)\right] \ge 1 - \delta'. \tag{22}$$

Then we can employ scaling techniques:

$$\sum_{j\neq i^*} T_j(n) \le \sum_{j\neq i^*} O\left(\frac{b_i \ln n}{\Delta_j^2 \delta'}\right) \le O\left(\frac{b_i \ln n}{\Delta_{\min}^2 \delta'}\right). \tag{23}$$

According to equation 23, if we need to derive the condition that satisfies equation 21, the inequality $\sum_{j\neq i^*} T_j(n) \ge O\left(\frac{b_i \ln n}{\Delta_{\min}^2 \delta'}\right)$ should be satisfied.

Then using the one-sided Hoeffding inequality, $\Pr[\mu^\star - \mu \le -\epsilon] \le \exp(-2T\epsilon^2)$, with the deviation threshold set to $\epsilon = \Delta_{\min}/2$. By requiring the probability of the empirical mean deviation event to be bounded by $c/t^2$ which ensures the summability of error probabilities via a convergent series, we obtain the constraint $\exp\left(-\frac{T_{i^*}(t)\Delta_{\min}^2}{2}\right) \le \frac{c}{t^2}$. Solving this inequality for the number of pulls $t$ yields $T_{i^\star}(t) \ge \frac{2\ln(t^2/c)}{\Delta_{\min}^2} = \frac{4\ln t - 2\ln c}{\Delta_{\min}^2}$, which simplifies asymptotically to $O\left(\frac{\ln t}{\Delta_{\min}^2}\right)$. Consequently, let $t = n$, the required sampling complexity for the optimal arm is lower-bounded by $T_{i^\star}(n) \ge O\left(\frac{\ln n}{\Delta_{\min}^2}\right)$. Then we can derive the sampling complexity:

$$n = \sum_{j\neq i^\star} T_j(n) + T_{i^\star}(n) \ge O\left(\frac{b_i \ln n}{\Delta_{\min}^2 \delta'}\right) + O\left(\frac{\ln n}{\Delta_{\min}^2}\right) = O\left(\frac{b_i \ln n}{\Delta_{\min}^2 \delta'}\right) \tag{24}$$

To solve equation 24, substitute the candidate $n^\star = A\ln A$ with $A = b_i/(\Delta_i^2\delta')$. Then $\ln n^\star = \ln(A\ln A) = \ln A + \ln\ln A \le 2\ln A$ for $A \ge e$, and the right-hand side of equation 24 becomes $O\left(A\ln n^\star\right) = K \cdot 2A\ln A = 2Kn^\star = O\left(n^\star\right)$, where $K$ is a constant. Hence the minimum $n$ satisfying equation 24 is $n^\star = O(A\ln A) = O\big(b_i \log(b_i/(\Delta_i^2\delta'))/\Delta_i^2\big)$. Setting $\Delta_i \ge w$ (Assumption 2) and $\delta' = \delta/N$ (so the union bound over $N$ sections gives $1-\delta$) yields the per-section rollout cost $T_i = O\big(b_i \log(b_i N/\delta)/w^2\big)$.

*Step 3 (Bound for Skills Reward).* Suppose at every section $i$ UCB1 identifies an $\epsilon_i$-optimal arm $e_i^\star$ with $r_i(e_i^\star) \ge r_i(e) - \epsilon_i$ for all $e \in \mathcal{E}_i$. Then for any skill $\mathcal{S} = (e_1, \ldots, e_N)$,

$$\sum_{i=1}^N r_i(e_i^\star) \;-\; \sum_{i=1}^N r_i(e_i) \;=\; \sum_{i=1}^N \big[r_i(e_i^\star) - r_i(e_i)\big] \;\ge\; -\sum_{i=1}^N \epsilon_i. \tag{25}$$

Choosing $\epsilon_i = \epsilon/N$ gives a per-skill gap of $\sum_i r_i$ bounded by $\epsilon$. Combined with Assumption 1's coupling error $\epsilon_{\max}$, we have:

$$\begin{aligned} \mathcal{R}(\mathcal{S}) - \mathcal{R}(\mathcal{S}^\star) - \sum_{i=1}^N \epsilon_i \;&\le \mathcal{R}(\mathcal{S}) - \mathcal{R}(\mathcal{S}^\star) + \sum_{i=1}^N \big[r_i(e_i^\star) - r_i(e_i)\big] \\ &= \big[\mathcal{R}(\mathcal{S}) - \sum_{i=1}^N r_i(e_i)\big] + \big[\mathcal{R}(\mathcal{S}^\star) - \sum_{i=1}^N r_i(e_i^\star)\big] \le \epsilon_{\max}, \end{aligned} \tag{26}$$

which means the joint reward satisfies $R(\mathcal{S}^\star) \geq R(\mathcal{S}) - \epsilon - \epsilon_{\max}$, i.e., the optimality gap is at most $\epsilon + \epsilon_{\max}$. Identifying each section to within $\epsilon/N$ requires $O\big(b_i \log(b_i N/\delta)/(\epsilon/N)^2\big)$ samples per section, and summing over the $N$ sections yields equation 18 with $\epsilon$ set to a constant multiple of $w$ (the per-section accuracy floor of Assumption 2). □

**Dependence on $N$.** The bound equation 18 is *linear* in $N$ (up to logarithmic factors), because $\sum_i b_i \leq N \cdot \max_i b_i$ and each section is solved independently. This is the structural payoff of MCETS's section-decomposable tree: the joint search is decomposed into $N$ one-dimensional bandits. Under the guidance of the Hierarchical Meta-Optimizer, the search space is reduced to a significantly smaller semantic space in the current task, which also emphasizes the importance and necessity of the meta-optimizer's guidance.

### D.5 Exponential Lower Bound for Joint-Blind Methods

To show the **necessity of a hierarchical meta-optimizer in the MCETS** that attributes failures to skills and performs edit actions, where sections or code blocks of skills are treated as bandits rather than the whole skill document, we provide a further discussion. The bound in equation 18 exploits the section-level decomposition of Assumption 1. To isolate the value of *decomposition itself* from the value of any particular algorithm, we ask the complementary question: how many rollouts does any algorithm need if it observes only the joint reward $R(\mathcal{S})$ (i.e., the outcome reward of a trajectory), and does not exploit the additive structure of the $r_i$? We call such algorithms *joint-blind*. This includes, for example, single-bandit search over $\mathcal{S} = \prod_i \mathcal{E}_i$ without using the section structure or any method that simply treats a skill as an indivisible string and edits it as a whole. Theorem 2 below shows that a joint-blind algorithm needs an exponential number of rollouts.

**Theorem 2** (Joint-blind methods need $\tilde{\Omega}(b^N)$ samples)**.** *Consider any algorithm $\mathcal{A}$ that observes only the joint reward $R(\mathcal{S})$ without access to the section-level decomposition of $r_i$. For any $\epsilon > 2\epsilon_{\max}$ and $\delta \in (0, 1/4)$ there exists a rewardfunction $R$ satisfying Assumptions 1 and 2, with $\epsilon_{\max}$ such that $\mathcal{A}$ requires at least*

$$N_{\text{joint}}(\epsilon, \delta) \;=\; \Omega\left(\frac{\prod_{i=1}^{N} b_i \;\log(1/\delta)}{(\epsilon - 2\epsilon_{\max})^2}\right) \tag{27}$$

*rollouts to identify an $\epsilon$-optimal skill with probability $1 - \delta$.*

*Proof.* A joint-blind algorithm must treat $\mathcal{S}$ as a single arm out of $K = |\mathcal{S}| = \prod_i b_i$ candidates. In this case, we equivalently transfer this problem to the same bandit setting in Theorem 1.

*Step 1 (construction).* For each section $i$, define $r_i$ so that one distinguished edit $e_i^\star \in \mathcal{E}_i$ achieves $r_i(e_i^\star) = 1$ and every other edit $e \in \mathcal{E}_i \setminus \{e_i^\star\}$ achieves $r_i(e) \leq 1 - w$, with $w$ the constant from Assumption 2. Define the joint reward as

$$R(\mathcal{S}) \;=\; \sum_{i=1}^{N} r_i(e_i) \;+\; \eta(\mathcal{S}), \qquad |\eta(\mathcal{S})| \leq \epsilon_{\max}. \tag{28}$$

where $\eta(\mathcal{S}$ denotes the potential interaction effects among different sections in skills. The optimal skill is $\mathcal{S}^\star = (e_1^\star, \ldots, e_N^\star)$, with $R(\mathcal{S}^\star) = N + \eta(\mathcal{S}^\star)$.

*Step 2 (effective gap).* Consider any skill $\mathcal{S} = (e_1, \ldots, e_N)$ that is not the optimum, i.e. it differs from $\mathcal{S}^\star$ in at least one section. Suppose $\mathcal{S}$ differs from $\mathcal{S}^\star$ in exactly $k \geq 1$ sections. Then

$$\sum_{i=1}^{N} r_i(e_i) \;\leq\; N - kw. \tag{29}$$

so $R(\mathcal{S}) \leq N - kw + \epsilon_{\max}$, while $R(\mathcal{S}^\star) \geq N - \epsilon_{\max}$, according to Assumption 2. The joint-reward gap is therefore

$$R(\mathcal{S}^\star) - R(\mathcal{S}) \;\geq\; (N - \epsilon_{\max}) - (N - kw + \epsilon_{\max}) \;=\; kw - 2\epsilon_{\max}. \tag{30}$$

For an $\epsilon$-suboptimal skill, one with $R(\mathcal{S}^\star) - R(\mathcal{S}) \le \epsilon$, the joint reward differs from the optimum by at most $\epsilon$, so the effective gap that a joint-blind algorithm must overcome is

$$\Delta \;=\; \epsilon - 2\epsilon_{\max}. \tag{31}$$

The bound is non-trivial only when $\epsilon > 2\epsilon_{\max}$, i.e. the target optimality exceeds the coupling-noise budget; otherwise $\eta$ alone could push the gap below zero, and no algorithm can guarantee $\epsilon$-optimality from finite rollouts.

*Step 3 (apply lower-bound theorem).* A joint-blind algorithm observes only the joint reward $R(\mathcal{S})$ and treats each $\mathcal{S} \in \mathcal{S}$ as an arm of a $K$-armed bandit with $K = |\mathcal{S}| = \prod_i b_i$. By Mannor & Tsitsiklis (2004) Theorem 1, any algorithm on a $K$-armed bandit with minimum gap $\Delta$ requires at least $\Omega(K \log(1/\delta)/\Delta^2)$ samples to identify the best arm with probability $1-\delta$. Substituting $K = \prod_i b_i$ and $\Delta = \epsilon - 2\epsilon_{\max}$ recovers equation 27, we have the lower bound of rollouts for joint-blind methods:

$$N_{\text{joint}}(\epsilon, \delta) \;=\; \Omega\left( \frac{\prod_{i=1}^{N} b_i \log(1/\delta)}{(\epsilon - 2\epsilon_{\max})^2} \right). \tag{32}$$

The theorem indicates the exponential complexity of the joint-blind methods, showing the necessity of a meta-optimizer that proposes fixed directions for skill patches for MCTS-based skill evolution. □

### D.6 Exponential MCETS Advantage

In this subsection, we provide a theoretical analysis of the superiority of the meta-optimizer in MCETS, especially for the case where the number of skill documents is large and the structure of skills is complex in a skill library.

**Theorem 3** (Exponential MCETS advantage in section count $N$)**.** *Under the assumptions of Theorems 1 and 2, and let $\epsilon_0$ satisfy: for any $\epsilon > 0$, $\epsilon_0 = \epsilon - 2\epsilon_{\max>0}$, then the ratio of joint-blind sample complexity to MCETS sample complexity is*

$$\frac{N_{\text{joint}}(\epsilon, \delta)}{N_{\text{MCETS}}(\epsilon, \delta)} \;=\; \Omega\left( \frac{w^2 \cdot \prod_i b_i \cdot \epsilon_{\max}^2 \cdot \log(1/\delta)}{\epsilon_0^2 \cdot N \cdot \max_i b_i \cdot \log(b_i N/\delta)} \right). \tag{33}$$

*When $\prod_i b_i$ grows geometrically with the number of sections $N$ (the typical case for skills with $b \ge 2$ edits per section), the ratio is exponential in $N$.*

*Proof.* Combine the upper bound equation 18 and the lower bound equation 27; the ratio is

$$\frac{\prod_i b_i \, \log(1/\delta)\, \epsilon_{\max}^2/\epsilon^2}{\sum_i b_i \, \log(b_i N/\delta)/w^2} \;\ge\; \frac{w^2 \prod_i b_i \, \epsilon_{\max}^2 \log(1/\delta)}{\epsilon^2 \cdot N \cdot \max_i b_i \cdot \log(b_i N/\delta)}.$$

For $b_i \ge 2$ the product $\prod_i b_i$ is at least $2^N$ while $\sum_i b_i$ is at most $N \max_i b_i$, so the ratio grows at least as fast as $c^N/(N \log b)$ for a constant $c \ge 2$ depending on the per-section edit sizes. □

## E Theoretical Analysis of the Back-Propagation Scheme

In this section, we present a full theoretical analysis of the linear back-propagation weight $w_k$ used in Section 2.3. The analysis is organized as follows: *Setup* (UCB selection at the root), *worst-case deception distribution* (Assumption 3 and its consequences), the main *monotonicity theorem* (Theorem 4) with its proof and accompanying remarks, the *deception-threshold corollary* (Corollary 1), and the *UCB regret consequence*. ***We show that using increasing weights for backpropagation for skills evolution can prevent the evolution from being trapped in suboptimal paths.***

### E.1 Setup: UCB Selection at the Root

Let the state $\sigma_t$ at shallow layers of the tree have $K$ child actions $a_1, \ldots, a_K$. Without loss of generality, we consider two of them that lead respectively to subtrees $\mathcal{T}_1$ (globally optimal) and $\mathcal{T}_2$ (deceptively locally optimal). After $N(\sigma_t)$ rollouts, the value of an edge is the *subtree-aggregated*

reward weighted sum defined in equation 10. Concretely, for subtree $\mathcal{T}_i$ with leaf set $\mathcal{L}_i$ and visit count $N(\sigma_t, a_i)$, the empirical Q-estimate of the node is

$$Q(\sigma_t, a_i) \;=\; \frac{1}{N(\sigma_t, a_i)} \sum_{v \in \mathcal{L}_i} w_{d(v)}\, \hat{r}_v, \qquad i \in \{1, \cdots, K\}, \tag{34}$$

where $d(v)$ is the depth of leaf $v$ and $\hat{r}_v$ is its empirical reward. We group leaves by depth to obtain a per-depth empirical mean $\hat{r}_{i,k}$ averaged over the $N_{i,k}$ visits to depth $k$ in $\mathcal{T}_i$, which gives the equivalent decomposition

$$Q(\sigma_t, a_i) \;=\; \sum_{k=1}^{L_i} w_k\, \hat{r}_{i,k}, \qquad \hat{r}_{i,k} \triangleq \frac{1}{N_{i,k}} \sum_{\substack{v \in \mathcal{L}_i \\ d(v)=k}} \hat{r}_v, \tag{35}$$

with $L_i$ the maximum depth of $\mathcal{T}_i$. The UCB1 selection rule equation 6 reduces, as $N(\sigma_t, a_i) \to \infty$ for both arms, to the sign of the Q-difference. Importantly, we construct *pseudo-nodes* to add nodes to shallower suboptimal subtrees and prune nodes of deeper suboptimal subtrees, making all the subtrees of equal size and comparable in value distributions at the same depth. Hence UCB asymptotically chooses $\mathcal{T}_1$ iff

$$Q(\sigma_t, a_1) - Q(\sigma_t, a_2) \;=\; \sum_{k=1}^{\max(L_1, L_2)} w_k \Delta_k \;>\; 0, \qquad \Delta_k \triangleq \hat{r}_{1,k} - \hat{r}_{2,k}, \tag{36}$$

with the convention that the value of the pseudo-node $\hat{r}_{i,k} = 0$ for $k > L_i$. The choice of weights therefore decides whether UCB can escape deceptive local optima.

### E.2 Worst-Case Deceptive Distribution

We define the *worst-case deception* $\mathcal{D}_{\text{worst}}$ as the reward-difference sequence that simultaneously satisfies three properties:

**Assumption 3** (Worst-case deception)**.** Considering the value distributions that are worst-case deceptive distributions among subtrees, they satisfy the following properties:

1. **Terminal truthfulness**: $\Delta_L > 0$ (the globally optimal subtree has strictly larger leaf value).
2. **Intermediate deception**: $\sum_{k=1}^{L-1} \Delta_k < 0$ (every prefix other than the leaf appears worse for $\mathcal{T}_1$).
3. **Tight margin**: $\sum_{k=1}^{L-1} \Delta_k + \Delta_L \approx 0$, i.e. the terminal advantage barely cancels the accumulated intermediate deficit.

Equivalently, write $\Delta_k = -\delta_k$ for $k < L$ with $\delta_k > 0$, and define the *deception ratio* $\rho \triangleq \sum_{k=1}^{L-1} \delta_k / \Delta_L > 1$.

This is precisely the setting of an extremely bad case: the better subtree loses on every intermediate step and wins *only* at the leaf.

Under Assumption 3, inequality equation 36 collapses to

$$w_L \Delta_L \;>\; \sum_{k=1}^{L-1} w_k \delta_k, \tag{37}$$

where $L = \max(L_1, L_2, \ldots, L_K)$ is the depth of the subtrees. The left-hand side is the terminal credit given to $\mathcal{T}_1$; the right-hand side is the penalty UCB would attach to $\mathcal{T}_1$ for its deceptive intermediates. Condition equation 37 is necessary and sufficient for UCB to select the globally optimal subtree.

### E.3 Monotonicity of the Back-Propagation Weight

**Theorem 4** (Monotonicity of the back-propagation weight is necessary under worst-case deception)**.** *Consider two subtrees of a node. Under Assumption 3, if the reward-difference sequence*

$\Delta_1 \le \Delta_2 \le \cdots \le \Delta_L$ *is non-decreasing, which represents the gap between a degrading evolution direction and an improving evolution direction, then* every *weight sequence* $\{w_k\}_{k=1}^{L}$ *satisfying equation 37 must be non-decreasing in* $k$*. This is necessary for the accumulated value gap between the optimal leaf path and the suboptimal leaf path to shrink and even reverse, so that the optimal path can overtake the suboptimal one to escape the worst-case deception .*

Note that this theorem is a worst-case necessity result, showing that SkillPilot can avoid the some extremely bad cases with higher tolerance.

*Proof.* We prove this theorem by a discrete swap argument on the objective $J(\mathbf{w}) \triangleq \sum_{k=1}^{L} w_k \Delta_k$.

Suppose $\{w_k\}$ is not non-decreasing. Then there exist indices $i < j$ with $w_i > w_j$. Swap the two weights and keep the rest fixed, defining $w'_i = w_j$, $w'_j = w_i$ and $w'_k = w_k$ for $k \notin \{i, j\}$. The change in $J$ is

$$\begin{aligned}\Delta J = \sum w'_k \Delta_k - \sum w_k \Delta_k &= (w_j \Delta_i + w_i \Delta_j) - (w_i \Delta_i + w_j \Delta_j) \\ &= (w_i - w_j)(\Delta_j - \Delta_i).\end{aligned}$$

Because $\Delta_j \ge \Delta_i$ (non-decreasing assumption) and $w_i > w_j$, both factors are non-negative, so $\Delta J \ge 0$. Equality holds only when $\Delta_j = \Delta_i$ or $w_i = w_j$, in which case the swap is a no-op.

The swap strictly increases $J$ whenever it acts on a strict inversion. Every weight sequence on a finite index set admits at most $\binom{L}{2}$ inversions, so a finite sequence of swaps eliminates all of them. Termination yields a non-decreasing sequence $\{\tilde{w}_k\}$ with $J(\tilde{\mathbf{w}}) \ge J(\mathbf{w})$. □

Among the class of monotonic sequence, the *linear* sequence $w_k = k$ is the canonical choice for two reasons. First, it is the *weakest* strictly-monotonic sequence, so it minimizes the Q-estimate variance $\sum_k w_k^2$ that appears in the denominator of the UCB regret bound equation 41 below. Second, under the additional mild assumption that $\Delta_k$ is *approximately linear in* $k$, the linear weights maximize $\sum_k w_k \Delta_k$ via the Cauchy–Schwarz inequality $\left(\sum_k k\Delta_k/L\right)^2 \le \left(\sum_k k^2/L^2\right)\left(\sum_k \Delta_k^2\right)$; any other monotonic pairing can only reduce the inner product by the rearrangement inequality. Concretely, for $\Delta_k = \alpha + \beta k$ with $\beta > 0$,

$$\sum_k k\Delta_k/L = \alpha \sum_k k/L + \beta \sum_k k^2/L^2 = \alpha \tfrac{L+1}{2L} + \beta \tfrac{(L+1)(2L+1)}{6L},$$

$$\sum_k w_k \Delta_k \le \sqrt{\sum_k w_k^2 \cdot \sum_k \Delta_k^2} \qquad \text{(Cauchy–Schwarz).}$$

Among all non-decreasing sequences with a given $\sum_k w_k^2$, the linear sequence minimizes the ratio between $\sum_k w_k^2$ and $(\sum_k w_k \Delta_k)^2$, hence maximizes the UCB signal-to-noise ratio equation 42 defined below.

**Corollary 1** (Linear scheme widens the tolerable deception threshold)**.** *In the worst-case distribution of Assumption 3 with uniform intermediate penalties* $\delta_k = \delta$ *for* $k < L$*, condition equation 37 reduces to*

$$\textit{uniform: } \rho < 1; \qquad \textit{linear: } \rho < 2 \qquad (\textit{for every } L \ge 2). \tag{38}$$

*Thus the linear scheme* widens *the maximum tolerable deception ratio by a factor of exactly* 2 *(over uniform weighting) for every* $L \ge 2$*.*

*Proof.* Under Assumption 3 with $\delta_k = \delta$ for $k = 1, \ldots, L-1$ and $\Delta_L > 0$, the deception ratio is

$$\rho = \frac{\sum_{k=1}^{L-1} \delta_k}{\Delta_L} = \frac{(L-1)\delta}{\Delta_L},$$

so $\Delta_L = (L-1)\delta/\rho$. Substituting into condition equation 37,

$$w_L \cdot \frac{(L-1)\delta}{\rho} > \delta \sum_{k=1}^{L-1} w_k \quad \Longleftrightarrow \quad \rho < \frac{w_L (L-1)}{\sum_{k=1}^{L-1} w_k}.$$

For *uniform* weights $w_k = 1$ we have $\sum_{k=1}^{L-1} w_k = L-1$ and $w_L = 1$, so $\rho < 1$. For *linear* weights $w_k = k$ we have $\sum_{k=1}^{L-1} w_k = (L-1)L/2$ and $w_L = L$, so

$$\rho \;<\; \frac{L(L-1)}{(L-1)L/2} \;=\; 2.$$

Hence the linear scheme widens the maximum tolerable deception ratio by a factor of exactly 2 over uniform weighting, for every $L \geq 2$. □

### E.4 UCB Regret Consequence

UCB1's selection rule at the root is

$$a^\star \;=\; \arg\max_{a \in \{a_1, a_2\}} \Big[\hat{V}(\sigma_t, a) + c\sqrt{\frac{\ln N(\sigma_t)}{N(\sigma_t, a)}}\Big], \tag{39}$$

where $\hat{V}(\sigma_t, a) = \frac{1}{N(\sigma_t,a)} \sum_{s: a_s = a} V(\mathcal{T}_{a,s})$ is the empirical mean of the back-propagated subtree value, and the exploration bonus depends on the per-action count $N(\sigma_t, a)$. UCB picks the wrong arm $a_2$ only if $\hat{V}(\sigma_t, a_1)$ underestimates the true value $V(\mathcal{T}_1) = \sum_k w_k r_k^{(1)}$ by at least half the gap $\Delta := V(\mathcal{T}_1) - V(\mathcal{T}_2) = \sum_k w_k \Delta_k$. Applying Hoeffding to the $N(\sigma_t, a_1)$ i.i.d. samples of $V(\mathcal{T}_1)$ with $\mathrm{Var}[V(\mathcal{T}_1)] = \sum_k w_k^2 \sigma^2$ gives

$$\Pr\big[\hat{V}(\sigma_t, a_1) \leq V(\mathcal{T}_1) - \Delta/2\big] \;\leq\; \exp\!\left(-\frac{2\,N(\sigma_t, a_1)\,(\Delta/2)^2}{\sum_k w_k^2 \sigma^2}\right) \;=\; \exp\!\left(-\frac{N(\sigma_t, a_1)\,\Delta^2}{2\,\sum_k w_k^2\,\sigma^2}\right). \tag{40}$$

A more careful analysis (using the per-action count and a sharper concentration over both arms' estimates) tightens the constant from 2 to 8, giving the displayed bound

$$\Pr(a_{\mathrm{UCB}} = a_2) \;\leq\; \exp\!\Big(-\frac{N(\sigma_t, a_1)\,\big(\sum_k w_k \Delta_k\big)^2}{8\,\sum_k w_k^2\,\sigma^2}\Big), \tag{41}$$

where $\sigma^2$ is the per-depth reward variance. The per-action count $N(\sigma_t, a_1)$ is essential: when $\mathcal{T}_1$ is underestimated by UCB (precisely the regime Theorem 4 addresses), $a_1$ is selected less often, so $N(\sigma_t, a_1)$ grows slowly while $N(\sigma_t, a_2)$ grows quickly. This *suppresses* $a_1$'s own exploration bonus, creating a self-reinforcing under-exploration that we call *visit-starvation*.

Define the *UCB effective signal-to-noise ratio*

$$\kappa(\mathbf{w}) \;\triangleq\; \frac{\big(\sum_k w_k \Delta_k\big)^2}{\sum_k w_k^2}. \tag{42}$$

Under the worst-case construction of Assumption 3 with uniform intermediate penalties $\delta_k = \delta$, the inner sum evaluates as

$$\sum_{k=1}^{L} w_k \Delta_k \;=\; w_L \Delta_L - \delta \sum_{k=1}^{L-1} w_k \;=\; \Delta_L - \frac{\delta}{2} \cdot \frac{2\sum_{k=1}^{L-1} w_k}{1}, \tag{43}$$

and using $\rho = (L-1)\delta/\Delta_L$ (i.e. $\delta(L-1) = \rho\,\Delta_L$),

$$\sum_{k=1}^{L} w_k \Delta_k \;=\; \Delta_L\Big(1 - \rho \cdot \frac{\sum_{k=1}^{L-1} w_k}{L-1}\Big). \tag{44}$$

For the two canonical weight sequences:

- *Linear* $w_k = k/L$: $\sum_{k=1}^{L-1} w_k = (L-1)/2$, $\sum_k w_k^2 = \Theta(L)$, so $\sum_k w_k \Delta_k = \Delta_L(1 - \rho/2)$.
- *Uniform* $w_k = 1$: $\sum_{k=1}^{L-1} w_k = L-1$, $\sum_k w_k^2 = L$, so $\sum_k w_k \Delta_k = \Delta_L(1-\rho)$.

Substituting into $\kappa$ gives

$$\kappa(\mathbf{w}_\ell) \;=\; \Theta\!\left(\frac{\Delta_L^2\,(1-\rho/2)^2}{L}\right), \qquad \kappa(\mathbf{w}_{\mathrm{unif}}) \;=\; \Theta\!\left(\frac{(1-\rho)^2\,\Delta_L^2}{L}\right). \tag{45}$$

The ratio $\kappa(\mathbf{w}_\ell)/\kappa(\mathbf{w}_{\mathrm{unif}}) = (1-\rho/2)^2/(1-\rho)^2$ is finite for $\rho$ bounded away from 1 but blows up as $\rho \to 1$, i.e. in the worst-case deception regime. Theorem 4 shows that the linear weight $\mathbf{w}_\ell$ maximises the numerator of $\kappa$, so $\kappa(\mathbf{w}_\ell) \geq \kappa(\mathbf{w})$ for any monotonic $\mathbf{w}$ with the same total mass.

**Regret consequence.** Substituting equation 45 into the standard UCB regret bound $R_T \leq 8\sigma^2\kappa(\mathbf{w})^{-1}\log T + O(1)$ gives

$$R_T(\mathbf{w}_\ell) \;\leq\; \frac{8\sigma^2\, L\, \log T}{\Theta\big(\Delta_L^2\,(1-\rho/2)^2\big)}, \qquad R_T(\mathbf{w}_{\text{unif}}) \;\leq\; \frac{8\sigma^2\, L\, \log T}{\Theta\big(\Delta_L^2\,(1-\rho)^2\big)}. \tag{46}$$

The linear scheme achieves a $\big((1-\rho)^2/(1-\rho/2)^2\big)$-factor improvement over uniform, which becomes unbounded as $\rho \to 1$ in the worst cases. However, according to equation 46, the adopted increasing weights improve the tolerance to $\rho \to 2$, preventing the unbounded regret. Actually, the empirical gap on real skill distributions is not so large, so the linear scheme with a slightly higher tolerance is a safe default in practice.

### E.5 Case Study: Navigating Skill Evolution via MCETS

In this section, we demonstrate a case where SkillPilot successfully and efficiently navigates the skills evolution from a suboptimal path to the optimal path. The details of the evolution tree and the skill modifications are shown in Figure 11.

As illustrated in the tree, traditional methods that edit skills along a *single adaptation trajectory* are easy to get trapped in suboptimal paths. For instance, the path $1 \to 2 \to 6 \to 7$ shows a promising intermediate state at node 6 (Reward=0.8), but eventually degrades to a reward of 0.4 at node 7 due to a failed decision from the LLM. Similarly, the path $1 \to 3 \to 5$ yields consistently poor performance, showing a very low-value path for skills evolution.

In contrast, our SkillPilot framework effectively navigates this vast search space by balancing exploration and exploitation. It evaluates the potential of different branches and can decisively abandon underperforming paths (e.g., nodes 3 and 7), and explore unvisited nodes (e.g., node 4). Specifically, even when the search encounters a temporary performance dip at node 4 (Reward=0.2) along the path $1 \to 2 \to 4$, the tree-search mechanism does not prematurely discard it. Instead, it continues to explore, ultimately reaching node 8 with a maximum Reward of 1.0.

This successful transition from node 4 to node 8 highlights the core advantage of SkillPilot: the accumulated skill advantages and decoupled error feedback from previous nodes are effectively transformed into *critical skill refinements*. By flexibly deciding *where* and *which way* to evolve, SkillPilot avoids the pitfalls of single-path adaptation, efficiently locating the optimal skill patches and substantially boosting the agent's in-context learning performance.

## F Limitations and Future Work

While SkillPilot demonstrates substantial improvements in joint skill optimization, it has certain limitations. Primarily, the MCETS framework is inherently dependent on the reward feedback obtained during the simulation phase to guide the tree search. In standard benchmarks, these rewards are well-defined; however, in open-ended real-world scenarios involving complex tasks, such as personal assistants, constructing a verifier to evaluate skill evolution performance is often inherently subjective. Consequently, there may exist gaps between the constructed verifier and the true evolution objectives, which could lead the search to overfit to constructed rewards.

To address these challenges, we identify several promising directions for future work. First, we plan to integrate explicit user instructions and human feedback directly into the MCETS search process. This integration would facilitate the real-time self-evolution of personal assistant agent skills, ensuring that the optimization trajectory remains tightly aligned with dynamic user preferences. Second, to enhance the efficiency of online evolution, parallelized MCTS algorithms can be explored, which can significantly accelerate the tree search and expansion stages when dealing with large-scale, continuously updating skill libraries. Finally, developing methods for constructing a unified evaluation environment and robust verifiers for skills is also a promising direction. By establishing standardized and objective assessment protocols, we can more precisely evaluate the correctness of the evolution direction, thereby bridging the gap between simulated rewards and actual task objectives.




Figure 11: Case of Skills Evolution in SkillsBench.