SELM: SELF-EVOLVING LIBRARY MANAGEMENT FOR MULTI-SKILL AGENT OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly use skills as a lightweight way to acquire task-specific knowledge. Recent work enables skills to evolve from task trajectories. However, these methods often lack systematic management of evolved skills, making it difficult for agents to reuse and accumulate skills across domains, thus limiting performance on subsequent tasks. To reliably accumulate, reuse, and maintain useful skills across tasks, we present SELM (Self-Evolving Library Management), a framework that formulates skill evolution as library-level management. SELM jointly manages skill retrieval, updating, and lifecycle by maintaining skill confidence and utility signals. It introduces utility-aware retrieval to select useful skills, a Map-Reduce framework to generate targeted updates from task-specific experience, and lifecycle controls to retain or remove skills based on their effectiveness. These mechanisms enable the skill library to continuously accumulate useful experience while preserving and pruning skills across tasks, improving agent performance on complex tasks across diverse domains. We evaluate SELM on four agent benchmarks covering diverse tasks and execution environments, and SELM consistently outperforms in most settings.

1 INTRODUCTION

Large language model (LLM) agents are increasingly used to solve complex tasks across different domains. To improve agents' behaviors on specific tasks, recent work introduced skills (Anthropic, 2025), which are pieces of text that can be plugged into the agent context to provide task-specific knowledge, procedures, or experience. This gives agents a simple way to gain specialized knowledge without additional model training.

Recent work has explored how skills can evolve as the agent runs. Early skills were mostly static: they were written by experts or saved from successful experiences and then kept unchanged (Zhao et al., 2024; Anthropic, 2025). More recent methods allow agents to update skills from their execution trajectories (Yang et al., 2026a; Alzubi et al., 2026; Liu et al., 2026; Zhang et al., 2026a; Zheng et al., 2025; Ni et al., 2026; Ma et al., 2026; Zhou et al., 2026; Zhang et al., 2026b). After running tasks, the agent can examine which skill worked or failed, then edit existing skills or create new ones. In this way, skills can change with the agent's experience and adapt to new tasks and environments.

However, although these evolution methods propose many strategies to evolve skills (Alzubi et al., 2026; Zhang et al., 2026a; Zheng et al., 2025; Zhou et al., 2026; Zhang et al., 2026b), few of them focus on how to manage them. A skill revision that best fits the current round can be harmful to later processes. For instance, a skill that is unused or unhelpful for the present task may be overwritten or discarded; once that change is committed, a later task can never recover the deleted skill. Meanwhile, failed or redundant skills that have remained in the system may still be loaded, giving the agent wrong instructions. The core problem is that existing methods always treat skill updates as separate, one-round edits, rather than managing skills as a dynamic, long-term library. Thus, a skill management library is needed to decide not only how a skill should change, but also which skills should be retrieved, which skills should be updated, and which skills should remain in the library.

To address these limitations, we propose SELM (Self-Evolving Library Management), a framework that manages skills as a library and jointly controls their retrieval, updates and lifecycle. By tracking

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

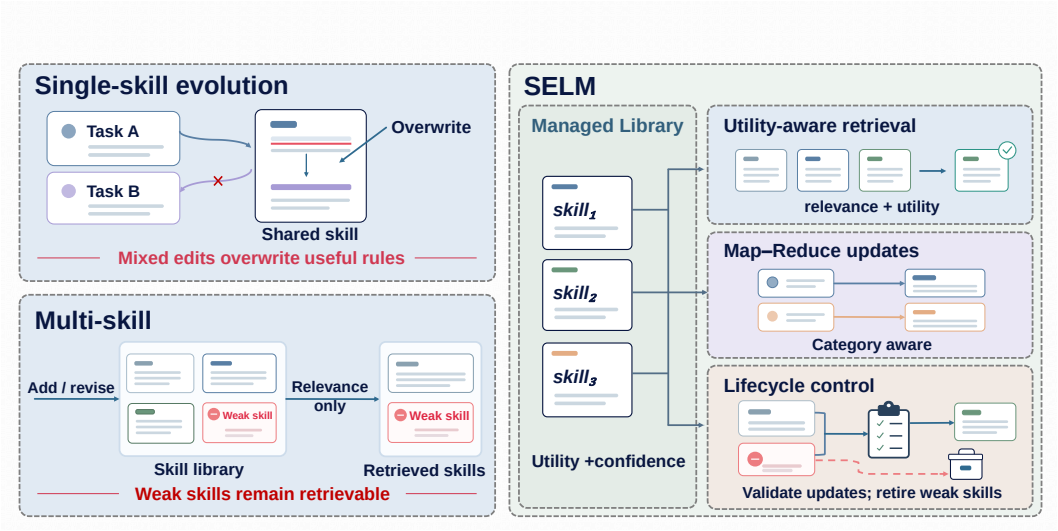

Figure 1: Prior skill-evolution methods can overwrite useful procedures or keep weak skills retrievable, causing task failures when experience from different tasks conflicts. SELM instead uses utility-aware retrieval, category-wise Map–Reduce updates, and lifecycle validation to preserve useful skills and retire ineffective ones, enabling more reliable task completion.

and leveraging skill confidence and utility, SELM enables the library to evolve with task experience and improves the performance of the agent across complex tasks in different domains.

Specifically, SELM introduces a **skill library** with explicit mechanisms for managing skill states. To dynamically evaluate skill usefulness, each skill is associated with a confidence score and a utility score. Confidence reflects how strongly the skill is supported by update operations, while utility reflects how useful it has been across tasks. These mechanisms provide a clear basis for deciding which skills should stay active, which skills should be removed, and which failed skills should not be added again. The library can thus keep a skill that is unused on the current task but still useful later, and drop one that keeps failing, instead of committing every local edit, enabling reliable, long-term skill management.

Second, SELM designs a **Map-Reduce framework** for skill updates. Applying every trajectory to all the skills will make experience from different types of task overwrites each other. SELM therefore groups execution trajectories by task category(Map) and analyzes each group separately, identifying which skills are affected and what changes are needed. It then combines these analyses into operations(Reduce), such as adding new skills, editing existing ones, and deleting those that no longer help. All these operations apply only to the relevant skills and unrelated skills are left unchanged, so a change that helps the current category won't erase skills required later, which enhances skill effectiveness.

Third, SELM proposes **utility-aware skill retrieval**. Existing methods mainly retrieve skills based on their semantic relevance to the current task (Yang et al., 2026b), which overlooks whether the skill has been useful in previous tasks. SELM leverages historical utility to enhance skill retrieval. It first selects skills from the same task category and then ranks them by their relevance to the current task and their past usefulness. This helps the agent distinguish skills that are relevant to task on hand and have proven useful through previous execution experience, thus enabling more effective skill selection.

Finally, we evaluate SELM on four agent benchmarks covering diverse execution environments and areas. SELM achieves state-of-the-art results among skill-evolution methods, outperforming both single-document optimization and prior multi-skill baselines without updating model parameters.

Our contributions are (1) introducing a **skill library** that tracks skill confidence and utility, managing skill validation, retention, and removal, enabling reliable skill management throughout the evolution process; (2) designing a **Map-Reduce framework** that analyzes trajectories by task category and

generates targeted operations, allowing the agent to update relevant skills based on task-specific experience; and (3) developing a **utility-aware retrieval** method that ranks skills using both task relevance and historical utility, enabling the agent to select skills that are both relevant and useful. Across our experiments, SELM outperforms existing skill optimization methods, demonstrating the effectiveness of the proposed framework.

2 RELATED WORK

Skill construction and evolution. Recent work has increasingly explored constructing and evolving textual skills from agent experience, moving beyond manually specified or static skill representations (Zhao et al., 2024; Anthropic, 2025). More broadly, approaches such as TextGrad (Yuksekgonul et al., 2025) and GEPA (Agrawal et al., 2026) optimize textual instructions through feedback without updating model parameters, while Dynamic Cheatsheet (Suzgun et al., 2026) and ACE (Zhang et al., 2026c) iteratively refine textual context from agent experience. Building on this paradigm, SkillOpt and SkillGrad optimize a shared skill document or skill package by repeatedly editing textual knowledge based on execution feedback (Yang et al., 2026a; Wang et al., 2026). Other approaches distill execution trajectories into standalone lessons, strategic principles, or verified skills, thereby converting individual experiences into reusable knowledge (Zhao et al., 2024; Wu et al., 2026; Ni et al., 2026; Ma et al., 2026; Qiu et al., 2026). Rather than maintaining a single shared artifact, SkillWeaver (Zheng et al., 2025), EvoSkill (Alzubi et al., 2026), SkillEvolver (Zhang et al., 2026a), Memento-Skills (Zhou et al., 2026), and CoEvoSkills (Zhang et al., 2026b) construct collections of specialized skills to capture different task types or capabilities. Complementary approaches construct skills from external domain resources or learn skill-related policies through interaction and co-training (Yang et al., 2026b; Liu et al., 2026; Xia et al., 2026; Jiang et al., 2026). Despite these advances, existing methods primarily focus on how skills are created or improved, treating skill evolution as a sequence of local construction or editing operations. This becomes limiting when multiple skills coexist: updates to a shared artifact can introduce conflicting changes, while independently accumulated skills provide limited mechanisms for determining which skills should be updated.

Skill library management. Early work such as Voyager (Wang et al., 2023) and ExpeL (Zhao et al., 2024) retrieves executable or experiential knowledge according to the relevance of the current task. Subsequent systems make skill selection more behavior-aware and introduce mechanisms for validating candidate skills or deciding whether updates should be merged, accepted, or discarded (Zhou et al., 2026; Yang et al., 2026a;b; Alzubi et al., 2026; Zhang et al., 2026b; Qiu et al., 2026). More recent work explicitly considers the maintenance of the skill library itself. SkillOps studies how an existing skill collection can be periodically reorganized to address issues that emerge as the library grows (Pu et al., 2026), while MUSE-Autoskill organizes skill creation, catalog retrieval, per-skill memory, and test-driven refinement into successive lifecycle stages (Lin et al., 2026). Lifelong evaluations further suggest that usage frequency alone is insufficient to characterize the utility of a skill, motivating more explicit utility-aware management (Zhang et al., 2026d). However, these approaches generally address retrieval, validation, update selection, or lifecycle maintenance as separate mechanisms, rather than jointly optimizing them as coupled decisions over a continually evolving skill library.

3 PRELIMINARY

This section introduces the basic concepts of skills, skill retrieval, and skill evolution.

Skill. The k-th skill is represented as

$$s_k = (n, d, c, t, \gamma), \tag{1}$$

where $n(s)$, $d(s)$, $c(s)$, $t(s)$, and $\gamma(s)$ denote the skill name, description, instruction content, trigger conditions, and task category, respectively.

As the number of skills grows, they form a skill library, which is a collection of reusable skills that the agent can draw on for different tasks. Formally,

$$\mathcal{L} = \{s_1, \ldots, s_{|\mathcal{L}|}\}. \tag{2}$$

Skill retrieval. The agent does not need to use the entire library for every task. Given a task x, a retrieval mechanism selects a subset of skills from the library to provide relevant instructions for the task. We denote the retrieved skill set as

$$\mathcal{S}(x) = \text{Retrieve}(\mathcal{L}, x), \qquad \mathcal{S}(x) \subseteq \mathcal{L}. \tag{3}$$

The selected skills are then provided to the agent as part of its context before task execution.

Evolving skills. With frozen target model M and current library, the agent runs on task x, and the resulting experience is written back so that the same library can improve over iterations. $r(\mathcal{L}, x)$ represents the score of library $\mathcal{L}$ on task x. Given training, selection, and test splits D_{tr}, D_{sel}, and D_{test}, evolution proposes libraries from D_{tr} and keeps the one that scores best on D_{sel}. Formally,

$$\mathcal{L}^* = \arg \max_{\mathcal{L} \in \mathcal{C}(D_{\text{tr}})} \frac{1}{|D_{\text{sel}}|} \sum_{x \in D_{\text{sel}}} r(\mathcal{L}, x), \qquad r(\mathcal{L}, x) \in [0, 1] \tag{4}$$

where $\mathcal{C}(D_{\text{tr}})$ is the set of libraries obtained by evolving on the training split. The score $r(\mathcal{L}, x)$ and the operators that map trajectories to library edits are instantiated in §4.

CRUD operations. CRUD is an operational concept in databases for managing persistent records. It stands for Create, Read, Update, and Delete. We migrate this concept to skill management. In this paper, the library operations are ADD, EDIT, UPVOTE, DOWNVOTE, and DELETE: ADD creates a new skill, EDIT revises an existing skill, UPVOTE and DOWNVOTE adjust its confidence, and DELETE removes it from the active library.

4 METHOD

SELM freezes the target model M and optimizes only the skill library. At each iteration, the agent retrieves a small set of skills from the current library, uses them to solve a batch of training tasks, and converts the resulting trajectories into library updates. The updated library is then evaluated on the selection set and retained only when it passes the validation gate. This iterative process jointly improves skill retrieval, skill updates, and library quality, with the overall procedure summarized in Algorithm 1 (Appendix A).

For each skill, SELM manages it through two pieces of information: confidence score and utility score. The confidence score $v(s)$ reflects how strongly a skill should be retained based on update operations, while the utility score $u(s)$ records how useful the skill has been across tasks. These two scores support different stages of the library evolution process: confidence is updated by UPVOTE/DOWNVOTE operations and used for lifecycle control, while utility is used to guide skill retrieval and updated from execution outcomes. Their corresponding update mechanisms are described in §4.1 and §4.2, respectively.

4.1 UTILITY-AWARE SKILL RETRIEVAL

SELM retrieves skills by jointly considering task category, task relevance, and historical utility. For each task x_i, let γ_i denote its task category. SELM first constructs a candidate set from the active skill set $\mathcal{S}_t$, retaining skills whose categories match γ_i or are marked as "general". It then ranks the candidates using a utility-aware retrieval score and selects the top-k skills for injection. Formally,

$$R(s, x_i) = (1 - w_u) R_{\text{rel}}(s, x_i) + w_u u(s), \tag{5}$$

where $R_{\text{rel}}(s, x_i)$ measures the relevance between skill s and task x_i using semantic and keyword-based matching, and $u(s)$ is the historical utility of the skill. The coefficient w_u controls the contribution of historical utility.

The top-k skills are selected for task x_i:

$$\mathcal{S}_t(x_i) = \text{top-}k \left(\{ s \in \mathcal{S}_t \mid \gamma(s) \in \{\gamma_i, \text{"general"}\} \}, R(s, x_i) \right). \tag{6}$$

The selected skills $\mathcal{S}_t(x_i)$ are injected into the agent context before execution. This retrieval strategy favors skills that are both relevant to the current task and useful based on previous execution results.

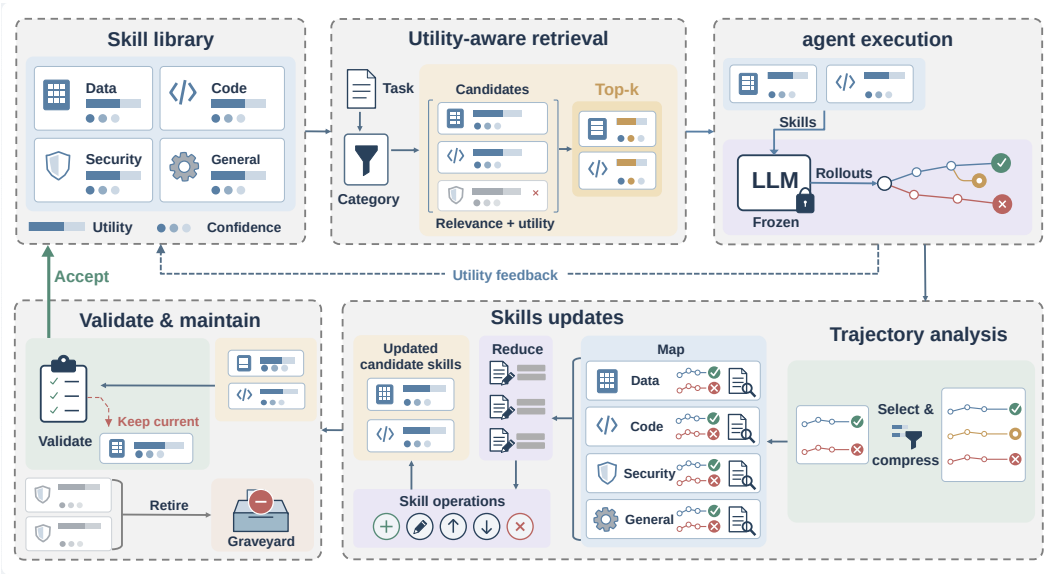

Figure 2: Overview of SELM. The frozen agent retrieves skills by relevance and utility, executes tasks, and turns trajectories into category-wise Map–Reduce updates. Candidate skills are validated on a held-out set; accepted skills remain in the library, and low-utility skills are retired to a graveyard that blocks their names from returning.

Utility Feedback. The utility score measures how useful a skill has been on the tasks that used it. Let $\mathcal{T}(s)$ denote the set of tasks whose retrieved skill set contains s. The average reward of s in the current update is

$$\bar{r}_s = \frac{1}{|\mathcal{T}(s)|} \sum_{x \in \mathcal{T}(s)} r(\mathcal{L}_t, x). \tag{7}$$

We update the utility using an exponential moving average (EMA)

$$u(s) \leftarrow (1-\alpha)u(s) + \alpha \bar{r}_s, \tag{8}$$

where α controls the contribution of new observations. The same utility score is then used by the retrieval mechanism in the next iteration. Thus, utility provides a feedback signal across successive retrieval rounds: execution outcomes update the utility of the skills that were used, and the resulting utility is subsequently incorporated into retrieval, reducing the selection of ineffective skills and improving agent performance across tasks.

4.2 SKILL LIBRARY EVOLUTION

After retrieving skills for each task, SELM runs the frozen target model and uses the resulting trajectories to update the skill library. The update process consists of four steps: rollout, trajectory analysis, CRUD synthesis, and candidate construction.

Rollout and Reward. For each training task x_i, the selected skills $\mathcal{S}_t(x_i)$ are injected into the agent context. Given the frozen target model M and execution harness h, the rollout produces a trajectory and task scores:

$$(\tau_i, r_{\text{hard},i}, r_{\text{soft},i}) = h\left(M, x_i, \mathcal{S}_t(x_i)\right), \tag{9}$$

where $r_{\text{hard},i} \in \{0,1\}$ indicates whether the task x_i is fully solved, and $r_{\text{soft},i} \in [0,1]$ measures partial progress in two ways: When the task x_i is judged by multiple checks, $r_{\text{soft},i} = n_{\text{pass},i}/n_{\text{total},i}$, and $r_{\text{hard},i} = 1$ only when every check passes. When a task is judged by comparing a prediction with a reference answer, $r_{\text{soft},i}$ is their token-level overlap and $r_{\text{hard},i}$ is exact match.

The reward used for library optimization is obtained from these task scores:

$$r_i = r(\mathcal{L}_t, x_i) = \mathcal{G}\left(r_{\text{hard},i}, r_{\text{soft},i}\right), \tag{10}$$

where

$$\mathcal{G}(r_{\text{hard}}, r_{\text{soft}}) = \begin{cases} r_{\text{hard}}, & \text{if metric} = \text{hard}, \\ r_{\text{soft}}, & \text{if metric} = \text{soft}, \\ (1-w)r_{\text{hard}} + w r_{\text{soft}}, w \in [0,1], & \text{if metric} = \text{mixed}. \end{cases} \tag{11}$$

Each rollout therefore yields two signals for library evolution. The trajectory τ_i is the execution evidence, including the agent's actions, tool calls, intermediate outputs, and errors. The scalar reward r_i shows how much of the task was solved. Later updates read τ_i to decide what to change, and use r_i to decide which trajectories are worth changing from.

Trajectory Selection. SELM prioritizes trajectories whose observed rewards differ substantially from the historical utility of the retrieved skills. For task x_i, we compute the average utility of its retrieved skills:

$$\bar{u}(\mathcal{S}_t(x_i)) = \frac{1}{|\mathcal{S}_t(x_i)|} \sum_{s \in \mathcal{S}_t(x_i)} u(s), \tag{12}$$

and define the diagnostic priority as

$$p_i = |r_i - \bar{u}(\mathcal{S}_t(x_i))| \,. \tag{13}$$

SELM selects the highest-priority trajectories for further analysis. Trajectories with large p_i provide stronger evidence that the retrieved skills may need to be updated. $\bar{u}(\mathcal{S}_t(x_i))$ is the library's current belief about those skills, and a large gap means the observed reward may contradict that belief. A much higher r_i shows a successful pattern that the stored utility under-credits, and a much lower r_i shows that the retrieved guidance failed on this task. A small gap means the rollout agrees with the existing record, so it adds little new information about what to change.

Map—Category-wise Analysis. The selected trajectories are partitioned by task category and analyzed independently. For each category, a `TrajectoryAnalyst` identifies successful patterns, failure patterns, and new insights from the corresponding trajectories. The analysis prevents experience from unrelated task categories from being directly mixed during skill updates.

Reduce—Skill update. Given the category-wise trajectory analyses and the current skill library, a `SkillSynthesizer` generates a bounded set of CRUD operations. The synthesizer considers the current skill content, confidence, utility, and category-specific execution feedback when deciding how to update the library.

Given a set of trajectories, SELM updates the library through the CRUD operations introduced in §3:

$$\mathcal{A} = \{\text{ADD}, \text{EDIT}, \text{UPVOTE}, \text{DOWNVOTE}, \text{DELETE}\}. \tag{14}$$

For an existing skill s, the confidence score is updated as

$$v(s) \leftarrow \begin{cases} v(s)+1, & \text{for UPVOTE}, \\ v(s)-1, & \text{for DOWNVOTE}. \end{cases} \tag{15}$$

The generated operations are applied to the relevant skills:

$$\mathcal{L}'_t = \text{Apply}\left(\mathcal{L}_t, \text{ops}_t\right), \qquad \text{ops}_t \subseteq \mathcal{A}. \tag{16}$$

When a category exhibits recurring failure patterns, the synthesizer can also create new skills that describe strategies for avoiding these failures. The resulting candidate library $\mathcal{L}'_t$ is then passed to the validation stage.

Multi-timescale Updates. Following prior work on iterative skill evolution (Yang et al., 2026a), SELM uses both fast and slow updates. Fast updates are performed after each training batch to incorporate recent trajectory feedback, while slow updates are performed less frequently across epochs to consolidate changes that persist over time. We also allow slow updates to produce meta skills, which summarize persistent cross-skill patterns identified across an epoch. This allows the library to respond to recent task feedback while reducing unnecessary updates caused by short-term fluctuations.

4.3 SKILL LIFECYCLE MANAGEMENT

SELM treats skill maintenance as an explicit lifecycle process, using validation, utility tracking, and retirement mechanisms to prevent ineffective skills from accumulating or re-entering the library. Throughout optimization, the active set $\mathcal{S}_t$ of $\mathcal{L}_t$ and the graveyard $\mathbf{G}_t$ obey three structural invariants:

$$|\mathcal{S}_t| \le M_{\max} \quad \text{(capacity)}, \tag{17}$$

$$\forall s_i, s_j \in \mathcal{S}_t,\ i \neq j \implies n(s_i) \neq n(s_j) \quad \text{(uniqueness)}, \tag{18}$$

$$\forall s \in \mathcal{S}_t,\ n(s) \notin \{n(g) \mid g \in \mathbf{G}_t\} \quad \text{(graveyard blocking)}. \tag{19}$$

Capacity bounds the store; uniqueness forbids duplicate names in $\mathcal{S}_t$; graveyard blocking forbids reintroducing a retired name.

Validation. A candidate update is not directly added to the active library. SELM first evaluates the candidate library $\mathcal{L}'_t$ on the selection set D_{sel} using the frozen target model and the same execution harness. Its validation score is

$$V(\mathcal{L}'_t) = \frac{1}{|D_{\text{sel}}|} \sum_{x \in D_{\text{sel}}} r(\mathcal{L}'_t, x). \tag{20}$$

The candidate library is retained only when it improves the validation performance over the current library. This validation gate prevents updates that appear reasonable from the trajectory analysis but reduce the actual task performance from being incorporated into the library.

Pruning and Graveyard. SELM sets a threshold τ_u for utility score. For skill s, if $u(s) < \tau_u$ or $v(s) < 0$, it will be retired from the active set $\mathcal{S}_t$ and moved to the graveyard $\mathbf{G}_t$. Retired skills are excluded from future retrieval and cannot simply be reintroduced under the same identity. This prevents ineffective skills from repeatedly entering and leaving the library and keeps the active library compact.

5 EXPERIMENTS

We evaluate SELM as a library-level optimizer for frozen agents. The experiments answer three questions. (i) Does a managed skill library improve over no skill, static skills, prompt evolution, single-document skill optimization, and recent multi-skill methods? (ii) Does that comparison hold across different models? (iii) Which library-management controls are required for the gains?

5.1 EXPERIMENTAL SETUP

Baselines. We compare against **No skill**; **Human skill** (an expert-written document per benchmark); **LLM skill** (a one-shot skill from Claude-Opus-4.6 that is never updated); prompt-evolution **GEPA** (Agrawal et al., 2026); skill-construction methods **SkillEvolver** (Zhang et al., 2026a), **Memento-Skills** (Zhou et al., 2026), and **Trace2Skill** (Ni et al., 2026); single-document **SkillOpt** (Yang et al., 2026a); and, under Claude Code, folder-based **EvoSkill** (Alzubi et al., 2026). EvoSkill is omitted from the no-harness block because it assumes a `SKILL.md` package layout. Qwen3-30B-A3B is evaluated only without a harness.

Benchmarks. We use four datasets: SkillsBench (Li et al., 2026), SpreadsheetBench (Ma et al., 2024), OfficeQA (Opsahl-Ong et al., 2026), and LiveMath (He et al., 2026). Detailed descriptions are in Appendix B.

Target models. We use Claude-Opus-4.6 (Anthropic, 2026), DeepSeek-V4-Pro (DeepSeek-AI, 2026), and Qwen3-30B-A3B (Yang et al., 2025) as frozen target models. Claude and DeepSeek are evaluated both without a harness and inside Claude Code (Anthropic, 2025); Qwen3-30B-A3B is evaluated only without a harness. No-harness runs prepend retrieved skills to the system prompt. Claude Code loads skills through the native agent workspace. All methods in a given cell use the same target model, harness, and scorer.

Hyperparameters. Hyperparameters are reported in Appendix C.

Table 1: Main results on SkillsBench, OfficeQA, SpreadsheetBench, and LiveMath. SkillsBench reports soft/hard accuracy (%); other rows are accuracy (%). The best cell in each row within a model block is bold and shaded purple; the second-best cell is shaded a lighter purple. Note: Qwen3-30B-A3B failed all the tasks on SkillsBench, so the Hard Acc is not included.

No harness

Benchmark		No skill	Human skill	LLM skill	GEPA	SkillEvolver	Memento-Skills	SkillOpt	Trace2Skill	SELM
Claude-Opus-4.6										
SkillsBench	Soft Acc	40.4	45.3	44.8	56.4	59.3	66.1	62.7	52.4	**69.8**
	Hard Acc	31.5	36.3	34.5	41.8	43.6	48.2	46.0	40.2	**50.7**
OfficeQA	Acc	45.3	49.7	47.5	53.1	54.6	59.5	56.4	51.2	**62.8**
SpreadsheetBench	Acc	44.3	46.1	45.8	49.6	52.5	54.3	51.3	48.3	**56.8**
LiveMath	Acc	35.2	36.8	37.4	41.5	50.8	56.0	56.9	46.5	**61.3**
DeepSeek-V4-Pro										
SkillsBench	Soft Acc	21.1	30.3	25.7	36.9	39.5	44.2	43.9	34.3	**45.5**
	Hard Acc	13.1	19.9	16.5	26.5	28.4	32.7	31.8	25.0	**34.2**
OfficeQA	Acc	33.2	36.6	35.4	41.2	42.5	46.8	44.3	40.7	**48.9**
SpreadsheetBench	Acc	32.9	33.6	34.9	37.8	38.6	41.2	39.7	38.1	**42.4**
LiveMath	Acc	27.4	28.6	28.1	31.2	34.8	37.5	36.2	32.0	**39.4**
Qwen3-30B-A3B										
SkillsBench	Soft Acc	16.4	17.5	16.9	17.8	18.5	19.8	19.2	17.9	**20.5**
OfficeQA	Acc	25.8	27.6	26.1	28.0	28.9	30.8	28.6	30.2	**32.5**
SpreadsheetBench	Acc	30.0	30.0	30.0	30.6	31.0	33.8	31.5	30.8	**36.4**
LiveMath	Acc	23.6	24.1	23.9	25.8	27.5	30.2	28.4	26.2	**33.1**

Claude Code harness

Benchmark		No skill	Human skill	LLM skill	GEPA	SkillEvolver	Memento-Skills	EvoSkill	SkillOpt	SELM
Claude-Opus-4.6										
SkillsBench	Soft Acc	51.4	61.1	59.2	72.5	76.8	81.2	79.5	80.2	**82.8**
	Hard Acc	40.0	46.6	43.9	52.9	56.5	59.8	59.0	58.3	**60.9**
OfficeQA	Acc	57.6	61.2	60.4	66.5	69.8	74.8	**75.4**	71.5	73.1
SpreadsheetBench	Acc	19.7	34.7	33.2	55.3	72.6	78.7	66.7	80.0	**81.5**
LiveMath	Acc	38.5	42.0	42.8	46.0	51.5	54.4	50.2	53.7	**57.8**
DeepSeek-V4-Pro										
SkillsBench	Soft Acc	28.8	34.0	32.6	42.6	46.3	50.1	49.0	50.4	**51.6**
	Hard Acc	19.4	22.1	20.5	27.8	29.5	33.3	30.9	33.8	**35.1**
OfficeQA	Acc	38.9	43.1	41.5	44.0	46.2	48.8	48.3	45.9	**50.0**
SpreadsheetBench	Acc	24.7	30.8	31.2	38.5	41.7	44.9	42.8	46.0	**46.9**
LiveMath	Acc	29.5	32.0	31.5	35.2	38.6	**41.5**	39.0	40.8	39.4

Table 2: Component ablation on SpreadsheetBench (Claude-Opus-4.6, no harness). w/o: without.

Setting	Acc (%)
Full (default)	**56.5**
w/o meta skill	53.1
w/o meta skill + slow update	42.7
w/o utility score	46.3
w/o utility-aware retrieval (w_u=0)	49.2
w/o validation gate	42.5
w/o graveyard	50.4

Table 3: Skill library evolution on SpreadsheetBench (Qwen3-30B-A3B, no harness). Init and epoch accuracies are on D_{sel}; Final is held-out test.

Epoch	Active	ADD	EDIT	DOWNVOTE	Graveyard	Acc. (%)
Init	5	–	–	–	0	17.5
1	10	5	0	3	0	17.5
2	11	2	0	4	–	30.0
Slow update	11	2	2	2	3	–
Final	11	9 (total)	2 (total)	9 (total)	3	36.43

5.2 MAIN RESULTS

Table 1 shows that SELM consistently improves agent performance across different benchmarks and target models. Compared with no-skill baseline and existing skill optimization methods, SELM achieves substantial gains in both no-harness and Claude Code settings, demonstrating the effectiveness of managing skills as a structured library rather than optimizing a single shared document. The improvements are also consistent across Claude-Opus-4.6, DeepSeek-V4-Pro, and Qwen3-30B-A3B, suggesting that the benefit of library-level optimization is not tied to a specific target model. Meanwhile, Static human-written skills and one-shot LLM skills provide only limited improvements, while trajectory-driven methods perform better but remain below SELM. This indicates that the main benefit comes from continuously managing which skills should be retrieved, updated, and retired. In particular, the advantage is more pronounced in heterogeneous settings such as SkillsBench, where task-specific feedback can conflict when compressed into a single skill document.

5.3 ABLATION STUDY

We ablate three aspects of SELM: library-management components, utility estimation, and optimizer–target model interaction. We first remove individual management mechanisms, including validation, utility-aware retrieval, meta skills, slow updates, and the graveyard. We then vary the EMA decay factor α to examine the sensitivity of utility estimation. Finally, we vary the optimizer and target models to study how optimizer capability affects the resulting skill library. Unless otherwise specified, the experiments follow the default configuration.

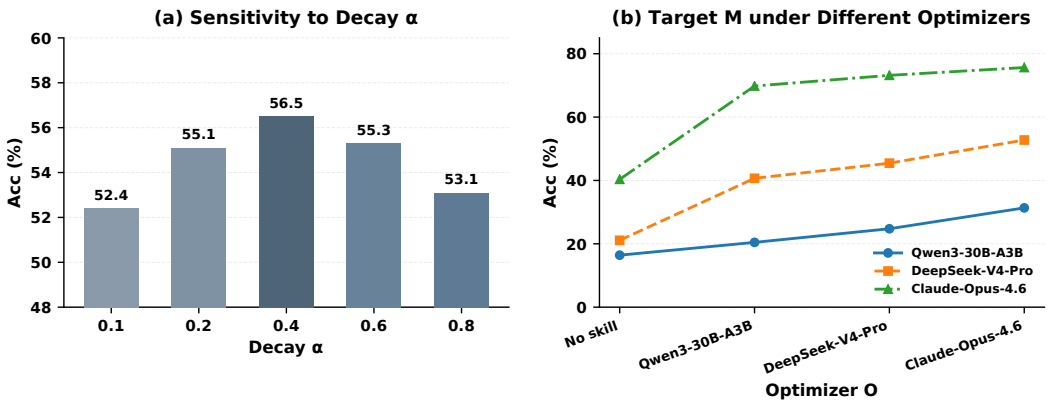

Figure 3: (a) Sensitivity to the EMA decay α on SpreadsheetBench (Claude-Opus-4.6, no harness); the marked default is α=0.4. (b) SkillsBench soft accuracy (%) for each target model M under different optimizers O, including the no-skill baseline (no harness).

Library-management controls are critical to the effectiveness of skill evolution. Table 2 shows that removing the validation gate causes the largest single drop, while removing the utility score and utility-aware retrieval reduces only 2.9 points. Removing both meta skills and slow updates also leads to a large degradation, compared with 53.1% when removing meta skills alone. The graveyard provides a smaller but consistent gain, with accuracy dropping to 50.4% without it. These results show that validation, utility-aware selection, and longitudinal library updates are important for preventing ineffective skill changes from accumulating.

SELM is relatively insensitive to the EMA decay factor around the default setting. As shown in Figure 3(a), α=0.4 achieves 56.5% accuracy, while α=0.2 and 0.6 obtain 55.1% and 55.3%, respectively. More extreme values, α=0.1 and 0.8, reduce accuracy to 52.4% and 53.1%. This indicates that the performance gain is not dependent on a narrowly tuned smoothing coefficient.

A stronger optimizer generally produces a more effective skill library, while weaker optimizers can still benefit stronger target models. Figure 3(b) shows that, with Qwen3-30B-A3B as the target, replacing the optimizer with DeepSeek-V4-Pro and Claude-Opus-4.6 increases soft accuracy from 20.45% to 24.75% and 31.34%, respectively. The same trend holds for DeepSeek-V4-Pro as the target. Meanwhile, using Qwen3-30B-A3B as the optimizer still improves Claude-Opus-4.6 from the no-skill baseline.

5.4 LIBRARY EVOLUTION ANALYSIS

Table 3 shows how the skill library changes during evolution on SpreadsheetBench with Qwen3-30B-A3B. The initial library contains 5 skills and achieves 17.5% accuracy on the selection set. In Epoch 1, five new skills are added, but the accuracy remains unchanged. This shows that simply adding more skills does not improve performance. In Epoch 2, only two new skills are added while four skills are downvoted, and the accuracy increases to 30.0%. The slow update then makes two edits, adds two meta skills, and downvotes two more skills. Three skills are eventually moved to the graveyard because their confidence becomes non-positive, and they are therefore excluded from future retrieval. The final library achieves 36.43% accuracy, compared with 30.0% for the no-skill baseline. These changes show that the library does not improve simply by growing in size; useful skills are added, while skills that repeatedly receive negative feedback are removed or revised.

6 CONCLUSION

We propose SELM, a framework that manages skills as a library. SELM tracks skill confidence and utility, validates skill updates, and retires failed skills. Through the Map-Reduce framework and utility-aware retrieval, SELM helps the agent select the most relevant and effective skills. Various experiments validate that SELM consistently improves frozen agents over prior skill-optimization methods and achieves the strongest performance in most settings.

AI USE STATEMENT

In this work, we used generative AI tools for drafting and polishing paper text, for retrieval and discovery of related work, and for research ideation and execution, including assistance in generating and running experimental code. We have reviewed all AI-assisted writing, claims, citations, and code. AI-assisted writing was reviewed and edited by the authors, information and references identified through AI-assisted retrieval were checked against the original sources, and AI-assisted experimental code was reviewed, run, and verified by the authors. We take responsibility for the final content of this work, including text, claims, citations, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies textual skill optimization for frozen LLM agents and does not involve human subjects, personally identifiable data, or the release of new datasets. Experiments use publicly available benchmarks (SkillsBench, SpreadsheetBench, OfficeQA, and LiveMath). The proposed methods update natural-language skill libraries rather than model weights, and they are intended for research on agent adaptation. We do not foresee immediate dual-use risks beyond those already associated with general-purpose LLM agents.

REPRODUCIBILITY STATEMENT

All hyperparameters are reported in Appendix C. The skill representation, library invariants, retrieval rule, Map-Reduce evolution procedure, CRUD operators, and maintenance mechanisms are specified in §4 and Algorithm 1. We will release the complete codebase and optimized skill libraries upon acceptance.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/0e9e708b6f48e14fd0ac29e167413f76-Paper-Conference.pdf.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems, 2026. URL https://arxiv.org/abs/2603.02766.

Anthropic. Equipping agents for the real world with agent skills, oct 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills. Engineering blog, 16 October 2025.

Anthropic. Claude Code, 2025. URL https://code.claude.com/docs/en/overview. Official documentation.

Anthropic. Introducing claude opus 4.6, feb 2026. URL https://www.anthropic.com/research/claude-opus-4-6.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Linyang He, Qiyao Yu, Hanze Dong, Baohao Liao, Xinxing Xu, Micah Goldblum, Jiang Bian, and Nima Mesgarani. Livemathematicianbench: A live benchmark for research-level mathematical reasoning with proof sketches, 2026. URL https://arxiv.org/abs/2604.01754.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. Xskill: Continual learning from experience and skills in multimodal agents, 2026. URL https://arxiv.org/abs/2603.12056.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026. URL https://arxiv.org/abs/2602.12670.

Huawei Lin, Peng Li, Jie Song, Fuxin Jiang, and Tieying Zhang. Muse-autoskill: Self-evolving agents via skill creation, memory, management, and evaluation, 2026. URL https://arxiv.org/abs/2605.27366.

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. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '26, pp. 4763–4768, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400725999. doi: 10.1145/3805712.3808466. URL https://doi.org/10.1145/3805712.3808466.

Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. Skillgen: Verified inference-time agent skill synthesis, 2026. URL https://arxiv.org/abs/2605.10999.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 94871–94908. Curran Associates, Inc., 2024. doi: 10.52202/079017-3007. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ac840df270ac537dd74530a15c332684-Paper-Datasets_and_Benchmarks_Track.pdf.

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, 2026. URL https://arxiv.org/abs/2603.25158.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, Michael Bendersky, Matei Zaharia, and Xing Chen. Officeqa pro: An enterprise benchmark for end-to-end grounded reasoning, 2026. URL https://arxiv.org/abs/2603.08655.

Hongji Pu, Xinyuan Song, and Liang Zhao. Skillops: Managing llm agent skill libraries as self-maintaining software ecosystems, 2026. URL https://arxiv.org/abs/2605.13716.

Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Liangyu Li, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. Autorefine: Compiling trajectories into validated typed agent artifacts, 2026. URL https://arxiv.org/abs/2601.22758.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, Rabat, Morocco,

2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.eacl-long.333. URL https://aclanthology.org/2026.eacl-long.333/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Hanyu Wang, Yifan Lan, Bochuan Cao, Lu Lin, and Jinghui Chen. Skillgrad: Optimizing agent skills like gradient descent, 2026. URL https://arxiv.org/abs/2605.27760.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. Evolver: Self-evolving llm agents through an experience-driven lifecycle. 2026. URL https://arxiv.org/abs/2510.16079.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL https://arxiv.org/abs/2602.08234.

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

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills, 2026a. URL https://arxiv.org/abs/2605.23904.

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

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639:609–616, 2025. doi: 10.1038/s41586-025-08661-4.

Genrui Zhang, Erle Zhu, Jinfeng Zhou, Caiyan Jia, and Hongning Wang. Skillevolver: Skill learning as a meta-skill, 2026a. URL https://arxiv.org/abs/2605.10500.

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

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026c. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/8a94ff6f922d995d7d3f4ebf4143e442-Paper-Conference.pdf.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. Skillflow:benchmarking lifelong skill discovery and evolution for autonomous agents, 2026d. URL https://arxiv.org/abs/2604.17308.

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. doi: 10.1609/aaai.v38i17.29936.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. Skillweaver: Web agents can self-improve by discovering and honing skills, 2025. URL https://arxiv.org/abs/2504.07079.

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

A Iterative Library Evolution Algorithm

Algorithm 1 summarizes the SELM loop in §4: utility-aware retrieval and rollout, diagnostic trajectory selection, Map–Reduce CRUD synthesis, validation gating, pruning to the graveyard, and slow updates.

B Benchmark Details

We evaluate on four agent benchmarks covering heterogeneous domains, procedural tool use, long-context document reasoning, and mathematical problem solving.

SkillsBench (Li et al., 2026): a multi-domain benchmark with 8 task categories (cybersecurity, data analysis, finance-economics, hardware engineering, natural science, social science, software engineering, and one general category). SkillsBench reports both hard accuracy and soft accuracy, corresponding to r_{hard} and r_{soft} in §4.1.

SpreadsheetBench (Ma et al., 2024): spreadsheet manipulation requiring multi-step formula generation and cell-level operations.

OfficeQA (Opsahl-Ong et al., 2026): a grounded reasoning benchmark built from U.S. Treasury Bulletins with a corpus of approximately 89,000 pages.

LiveMath (He et al., 2026): LiveMathematicianBench(LiveMath), a live mathematician-level reasoning benchmark, abbreviated LiveMath in tables.

The last three benchmarks report a single accuracy.

C Hyperparameter Summary

D Skill Library Example

Figures 4–7 reproduce four skills from the optimized SpreadsheetBench library (Qwen3-30B-A3B, no harness; 11 active skills, test accuracy 36.43%), one from each remaining category. Text is the full library snapshot, not a truncation.

Skill: `output_requirements`

Category. `general` **Confidence.** 4 **Utility.** 0.43
Description. Output Requirements.
Trigger conditions. None; injected whenever the skill is retrieved.
Content.

- Save the result to `OUTPUT_PATH`.
- Do not hardcode row counts or column letters — iterate over actual rows in the workbook.
- Preserve sheets and cells not mentioned in the instruction.

Table 4: Complete hyperparameter summary with default values.

Hyperparameter	Description	Default
$M_{\max}$	Maximum library capacity	30
w_u	Utility weight in retrieval scoring	0.6
w	soft score weight in reward calculating	0.5
α	EMA decay factor for utility update	0.4
τ_{soft}	Soft threshold for success classification	0.3
$\lvert\mathbf{G}\rvert_{\max}$	Maximum graveyard capacity	50
B_{ADD}	Max ADD operations per evolution step	5
B_{EDIT}	Max EDIT operations per evolution step	3
K	Number of skills retrieved per task	3
B	Trajectory selection budget	50
N_{slow}	Slow update sample size	20
τ_u	threshold for utility score	0.05
Injection budget	Max characters for skill injection	2000
Fail compression	Max chars for failed trajectories	2000
Success compression	Max chars for successful trajectories	400
Map parallelism	Max parallel TrajectoryAnalyst workers	16

Figure 4: A general skill from the optimized SpreadsheetBench library.

Skill: `dynamic_date_handling`

Category. `cell_level` **Confidence.** 2 **Utility.** 0.44
Description. Handle dynamic date ranges and exclude weekends/holidays in formulas.
Trigger conditions. When tasks involve date-range calculations with dynamic inputs or require exclusion of non-business days.
Content.

1. Use `DATE` functions to define start and end dates.
2. Use `NETWORKDAYS` or custom logic to exclude weekends.
3. Include holiday lists if provided.
4. Use `IF` statements to validate date ranges.
5. Test with sample data to ensure accuracy.

Figure 5: A cell-level skill from the optimized SpreadsheetBench library.

Skill: `inter_sheet_data_matching`

Category. `sheet_level` **Confidence.** 2 **Utility.** 0.44
Description. Handle data matching and manipulation across multiple sheets.
Trigger conditions. When tasks require data from multiple sheets to be compared, combined, or processed.
Content.

1. Use VBA to reference data from other sheets.
2. Use loops and conditionals to match data.
3. Use error handling to prevent runtime issues.
4. Test with different data structures.
5. Use message boxes to provide feedback on results.

Figure 6: A sheet-level skill from the optimized SpreadsheetBench library.

Skill: `slow_update_epoch_02`

Category. `meta` **Confidence.** 3 **Utility.** 0.70
Description. Epoch-level guidance from epoch 2 slow update.
Trigger conditions. None; produced by the epoch-level slow update and managed by the same utility and lifecycle rules as ordinary skills.
Content.

- **ADD**: `meta_insight_epoch_2` — Identified patterns of persistent failures and regressions, highlighting the need for better handling of dynamic data, text parsing, and formula execution in VBA.
  Content: Focus on improving dynamic data handling, text parsing, and formula execution. Address issues with date handling, string matching, and VBA macro behavior. Prioritize skills that were downvoted or showed persistent failure.
- **EDIT**: `dynamic_column_handling`
- **EDIT**: `advanced_text_parsing_in_vba`
- **DOWNVOTE**: `text_matching_strategies` (Showed persistent failure in text comparison and filtering tasks, indicating a need for re-evaluation of its implementation and use cases.)
- **DOWNVOTE**: `vba_data_preservation` (Caused regression in data preservation tasks, suggesting potential issues with how it handles macro execution and data integrity.)

Figure 7: A meta skill produced by the epoch-level slow update.

E Limitations and Future Work

Semantic deduplication. The graveyard mechanism operates at the name level (exact string match), preventing exact-name oscillation but not semantically equivalent skills with different names from coexisting. Embedding-based similarity detection would be a natural extension.
Cross-environment transfer. A skill library optimized on one benchmark could potentially warm-start optimization on another, analogous to transfer learning in neural networks. We leave exploration of cross-benchmark transfer to future work.
Scalability. While we demonstrate effectiveness with up to 30 skills across 8 categories, scaling to hundreds of categories or thousands of skills would require more sophisticated indexing (e.g., embedding-based retrieval) and potentially hierarchical library organization.
Optimizer model dependence. The quality of evolution depends on the optimizer LLM's ability to analyze trajectories and synthesize CRUD operations. Weaker optimizer models may produce lower-quality operations, and the interaction between optimizer capability and library quality deserves systematic study.

Algorithm 1 SELM — Iterative Library Evolution

Require: Frozen target model M, execution harness h, training split D_{tr}, selection split D_{sel}, epochs E, retrieval budget K, trajectory budget B, EMA coefficient α, utility weight w_u, retirement threshold τ_u, maximum capacity $M_{\max}$

$L_0 \leftarrow$ initial skill library ▷ empty or seed skills
$G_0 \leftarrow \emptyset$ ▷ graveyard starts empty
$L_{\text{best}} \leftarrow L_0$ ▷ snapshot kept for export
$V_{\text{best}} \leftarrow V(L_0)$ ▷ score on D_{sel}
for $e = 1, \ldots, E$ **do**
  for each training batch $\mathcal{B} \subset D_{\text{tr}}$ **do**
    ▷ one fast update
    $S_t \leftarrow \{s \in L_t : n(s) \notin \{n(g) \mid g \in G_t\}\}$ ▷ block retired names
    for each task $x_i \in \mathcal{B}$ **do**
      Let γ_i be the category of x_i
      $S_t(x_i) \leftarrow \text{TopK}\big(\{s \in S_t : \gamma(s) \in \{\gamma_i, \text{general}\}\}, R(s, x_i), K\big)$ ▷ utility-aware retrieval
      $(\tau_i, r_{\text{hard},i}, r_{\text{soft},i}) \leftarrow h(M, x_i, S_t(x_i))$ ▷ rollout
      $r_i \leftarrow \mathcal{G}(r_{\text{hard},i}, r_{\text{soft},i})$ ▷ scalar reward
      $p_i \leftarrow \big|r_i - \bar{u}(S_t(x_i))\big|$ ▷ gap from stored utility
    end for
    Select the top-B trajectories according to p_i ▷ most informative traces
    for each skill s used in the current batch **do**
      $u(s) \leftarrow (1-\alpha)u(s) + \alpha \bar{r}_s$ ▷ EMA utility
    end for
    Partition selected trajectories by task category ▷ Map
    for each category c **do**
      $a_c \leftarrow \text{TrajectoryAnalyst}(\text{trajectories of } c, L_t)$ ▷ success, failure, insights
    end for
    $\text{ops}_t \leftarrow \text{SkillSynthesizer}(\{a_c\}, L_t)$ ▷ Reduce; operation budgets
    $L'_t \leftarrow \text{Apply}(L_t, \text{ops}_t, G_t)$ ▷ candidate library
    Evaluate $V(L'_t) = \frac{1}{|D_{\text{sel}}|} \sum_{x \in D_{\text{sel}}} r(L'_t, x)$ ▷ validation gate
    if $V(L'_t) > V(L_t)$ **then**
      $L_t \leftarrow L'_t$ ▷ accept only if improved
    end if
    if $V(L_t) > V_{\text{best}}$ **then**
      $L_{\text{best}} \leftarrow L_t$ ▷ keep the best snapshot
      $V_{\text{best}} \leftarrow V(L_t)$
    end if
  end for
  if slow update is scheduled at epoch e **then**
    ▷ less frequent than fast updates
    Generate additional CRUD operations ▷ consolidate persistent trends
    $L'_t \leftarrow \text{Apply}(L_t, \text{ops}_t^{\text{slow}}, G_t)$
    Validate L'_t on D_{sel} ▷ same gate as the fast update
    if $V(L'_t) > V(L_t)$ **then**
      $L_t \leftarrow L'_t$
    end if
    if $V(L_t) > V_{\text{best}}$ **then**
      $L_{\text{best}} \leftarrow L_t$
      $V_{\text{best}} \leftarrow V(L_t)$
    end if
  end if
  Lifecycle pruning:
  for each $s \in L_t$ **do**
    if $u(s) < \tau_u$ **or** $v(s) \leq 0$ **then**
      ▷ low utility or non-positive confidence
      $L_t \leftarrow L_t \setminus \{s\}$
      $G_t \leftarrow G_t \cup \{s\}$ ▷ retire; name cannot return
    end if
  end for
  Enforce $|G_t| \leq |G|_{\max}$ ▷ graveyard capacity
  Enforce $|S_t| \leq M_{\max}$, name uniqueness, and graveyard blocking
end for
return L_{best} ▷ not necessarily the last L_t