SKILLVINE: AGENT SKILL EVOLUTION VIA BRANCHING EXPLORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills encapsulate reusable procedural knowledge that enables LLM agents to perform tasks, and they can be improved automatically using trajectories from interactions with the environment. This is the classic problem of skill evolution. Existing approaches predominately follow a linear evolution paradigm, in which updates are sequentially applied to the latest skill-library version. As a result, they inevitably fall into local optima, leaving many promising evolution paths unexplored. We propose SkillVine, an automatic skill-evolution framework that formulates skill evolution as a graph search problem and employs a branching exploration strategy. Equipped with a trunk-branch collaborative searching mechanism, an intelligent parent-node selector, and an adaptive-granularity update rule, SkillVine achieves a balance between exploration and exploitation. We evaluate SkillVine on 5 benchmarks with two LLMs. Results show that SkillVine discovers better skill-library versions along branches than along the linear trunk and achieves the best test performance in nine of ten benchmark–model combinations.

1 INTRODUCTION

Large Language Model (LLM) agents are increasingly used in software engineering (Yang et al., 2024; Jimenez et al., 2024; Qian et al., 2024), automated scientific research (Team et al., 2025; OpenAI, 2025b; Lu et al., 2024; Chan et al., 2025), web interaction (Google DeepMind, 2025; OpenAI, 2025a; Zhou et al., 2024; Yao et al., 2022), and embodied intelligence (Bolton et al., 2025; Wang et al., 2023). These applications require both general reasoning and specialized knowledge, yet putting extensive procedural knowledge directly in prompts is constrained by fixed context windows. Agent Skills (Zhang et al., 2025) address this limitation by packaging reusable procedures as modular, dynamically loadable executable units. However, static skill libraries cannot readily adapt to changing environments or tasks beyond their initial coverage, motivating research on self-evolving agents (Fang et al., 2025; Li, 2026) and automatic skill acquisition from interaction trajectories (Ni et al., 2026; Yang et al., 2026b; Pan et al., 2026). Yet simply adding more skills does not always improve performance: expanding a library from a small collection to 202 skills can reduce performance by up to 21%, with skill shadowing, in which agents increasingly select inappropriate skills as the library grows, accounting for up to 68% of this degradation (Song & Wei, 2026). Lifelong skill evolution must therefore encompass not only acquisition but also systematic refinement, pruning, and adaptation to sustain agent competence and safety in non-stationary environments.

Recent work has moved beyond one-time skill generation toward lifelong skill evolution. Existing approaches use reinforcement learning or model self-reflection to optimize either the agent's parameters (Wu et al., 2025; Xia et al., 2026) or its external skill library (Ouyang et al., 2026; Yang et al., 2026a; EvolvingLMMs-Lab, 2026). Most approaches, however, follow a **linear evolution paradigm** in which every update is applied sequentially to the latest skill library state. This paradigm introduces three limitations. First, as the skill library evolves along a single forward path, useful content from earlier versions may be overwritten, especially when new tasks differ greatly from previous ones. Second, each update builds sequentially on all previous updates, and some updates may lead evolution into a local optimum. Without version history, the system cannot undo these updates and explore alternative paths. For example, an update may add a rule that helps with current tasks but limits performance on later tasks. Under a linear evolution paradigm, the system cannot return to the version before this rule was added and explore a different path. Third, the linear paradigm forces the skill library to evolve along a single path, preventing parallel exploration and

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.

systematic comparison of different skill variants' performance. To overcome these limitations, we propose **SkillVine**, a framework that enables agent skill evolution through **branching exploration**.

To enable branching exploration, SkillVine organizes skill-library versions into a **version graph** that preserves the complete content of each version. Each node stores a skill-library version and its validation results, while directed edges record parent-child relationships between nodes, allowing the system to return to any earlier versions and explore new paths from them. SkillVine introduces three key mechanisms. First, it uses a **trunk-branch update strategy**. The trunk attempts an incremental update on every training batch, allowing useful updates to accumulate continuously along it. Branches grow from historical versions to explore alternative paths that provide capabilities the trunk lacks. Second, SkillVine uses a **complementarity-aware node selector** to choose historical skill-library versions as starting points for branch evolution, favoring those whose capabilities complement the trunk. It scores each version based on its validation accuracy, how many candidate child versions it has produced, and how its capabilities complement the trunk. Third, it uses **adaptive granularity** for branch updates to maintain an appropriate optimization step size based on whether the trunk accepts its update. This reduces the risk that noisy or misleading information in a training batch leads to harmful skill updates. Overall, the design of SkillVine automatically balances performance, exploration, and capability complementarity within a limited optimization budget.

We evaluate SkillVine with two models on five benchmarks covering spreadsheet manipulation, mathematical reasoning, and question answering. SkillVine outperforms six baselines in nine of ten model–benchmark combinations. With DeepSeek-V4-flash, it leads on all five benchmarks. Moreover, every best final skill library is discovered through a branch rather than the trunk, demonstrating the value of branching beyond linear evolution.

Overall, our main contributions can be summarized as follows:

- We propose SkillVine, a framework for agent skill evolution through branching exploration. It preserves historical skill-library versions and combines continuous improvement along a trunk with exploration of alternative paths through branches.
- We design a complementarity-aware node selector and an adaptive branch update strategy. The selector selects historical versions by jointly considering their performance, how often they have been expanded, and how their capabilities complement the trunk, while the update strategy adjusts branch update granularity based on the trunk's update outcome to help filter out harmful updates.
- We conduct extensive experiments on five benchmarks with two base LLMs. SkillVine achieves the best test performance in nine of ten benchmark–model combinations. Analysis shows that branches discover better skill library versions than those found along the trunk.

2 Related Work

2.1 Lifelong Evolution of Skills

The lifelong management of agent skills has recently gained traction as a pivotal research frontier within self-evolving agentic systems. Survey Li (2026) formalizes this domain through an eight-stage lifecycle, spanning evidence collection, proposal, validation, organization, retrieval, maintenance, distillation, and governance; and a ten-operator taxonomy (e.g., Add, Refine, Merge, Prune) designed to systematically govern skill trajectories. A growing body of contemporary literature operationalizes this conceptual architecture across specific domains, such as automatic skill curation (Ni et al., 2026; Yang et al., 2026b; Pan et al., 2026; Lin et al., 2026; Ouyang et al., 2026; Xia et al., 2026), test-time skill optimization (Yang et al., 2026a; EvolvingLMMs-Lab, 2026), and continuous lifelong management (Mao et al., 2026; Cai et al., 2026; Huang et al., 2026; Zhang et al., 2026). Collectively, these contributions signal a paradigm shift in the field: moving beyond naive skill storage toward scalable, verifiable, and safe lifelong evolution.

2.2 Branching Search and Exploration

Branching search has long been used to balance exploration and exploitation in large decision spaces. Monte Carlo Tree Search (MCTS) and its UCT variant (Kocsis & Szepesvári, 2006; Browne et al., 2012) perform value-guided expansion over search trees. Go-Explore (Ecoffet et al., 2019) explic-

itly archives promising states, returns to them, and then explores further. Population-Based Training (Jaderberg et al., 2017) periodically copies high-performing training states and perturbs them to continue exploration. Recent LLM agents have also incorporated branching search to improve reasoning, acting, and tool-use trajectories. LATS (Zhou et al., 2023) integrates MCTS with LLM-powered agents, value functions, and self-reflection, while subsequent work applies tree search to web agents (Koh et al., 2024) and agentic AutoML (Liang et al., 2026). In parallel, MCTS has also been used to synthesize reasoning trajectories or process supervision signals for LLM training (Luo et al., 2024; Guan et al., 2025). Unlike these methods, which branch over action, reasoning, or tool-use trajectories, our method branches over skill-library versions. Each node represents a complete skill-bank snapshot, and each edge corresponds to a parent-child relationship.

3 SKILLVINE

This section formulates skill library evolution as a budget-constrained optimization problem and introduces SkillVine, which performs branching exploration over skill-library versions using a recoverable version graph.

3.1 PROBLEM FORMULATION

We formulate skill evolution as a sequential optimization problem that maximizes agent performance on a validation distribution subject to a fixed budget. Let $S_v = \{s_{v,1}, \ldots, s_{v,n_v}\} \in \mathcal{S}$ denote the skill library at version node v, where n_v is the number of skills in that version. Each skill $s_{v,i}$ is a reusable procedural unit (e.g., a guideline or executable script) that an LLM agent π can use to complete a task. At each evolution step t, the agent interacts with a stream of tasks sampled from a distribution $\mathcal{D}_t$, generating trajectories $T_t = \{\tau_{t,1}, \ldots, \tau_{t,m}\} \in \mathcal{T}$. Each trajectory $\tau_{t,j} = (x_{t,j}, a_{t,j}, y_{t,j})$ comprises the input prompts $x_{t,j}$, the executed action sequence $a_{t,j}$ (including skill invocations), and the outputs $y_{t,j}$. These trajectories are then aggregated and summarized to produce candidate skill updates (i.e., additions, refinements, or deletions). Formally, an update operator $U : \mathcal{S} \times \mathcal{T} \to \mathcal{S}$ maps a selected skill library version S_v and the interaction trajectories $\mathcal{T}_t$ collected with that version at training step t to a candidate child version: $S_{v'} = U(S_v, \mathcal{T}_t)$.

The objective of skill evolution is to identify the optimal skill library S^* that maximizes expected accuracy on a validation distribution, subject to a constrained training budget. For each candidate update generated from a task stream, we make a binary decision: either discard the update or apply it to a specific skill library version, thereby systematically exploring alternative evolutionary paths. Let $\mathcal{V}_B \subseteq \mathcal{S}$ denote the set of skill library versions explored within training budget B. The corresponding optimization problem is formally defined as:

$$S^* = \arg\max_{S \in \mathcal{V}_B} \mathrm{Acc}(\pi, S; \mathcal{D}_{\mathrm{val}}), \tag{1}$$

where $\mathrm{Acc}(\pi, S; \mathcal{D}_{\mathrm{val}})$ denotes the expected accuracy of agent π equipped with skill library S on the validation distribution $\mathcal{D}_{\mathrm{val}}$.

3.2 SKILLVINE CORE CONCEPT

As shown in Figure 1, instead of following a single linear patching evolution path, SkillVine organizes all skill library versions into a version graph and adopts a branching and reversible strategy for skill evolution. The key concept lies in a cooperative trunk-and-branch update as follows:

Trunk as a Continuous Batch-Covering Path. Training data are fed batch by batch. To maintain a continuous optimization path over this stream, SkillVine keeps a **trunk** evolution path that is updated on every training batch. The trunk can see every single batch and accumulates long-horizon information. It also provides a reference for selecting branch nodes and deciding how to update them. This avoids repeatedly branching from early versions, which limits the accumulation of improvements over time. In contrast, **branches** grow from selected historical checkpoints. Since a branch node is not necessarily selected at every training step, each branch path uses only a subset of the training batches. Thus, the trunk tracks the cumulative effect of the full batch sequence, while branches explore alternative evolution paths using a subset of training batches.

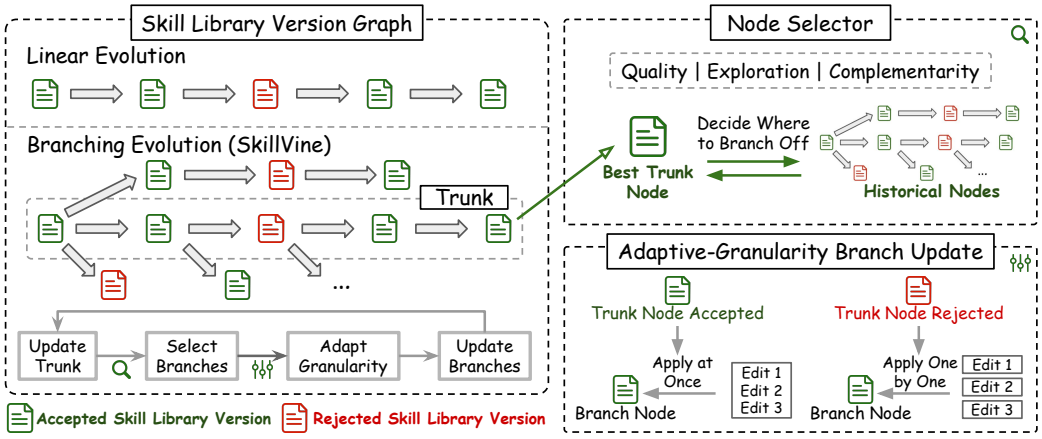


Figure 1: Overview of SkillVine. SkillVine organizes skill library versions into a graph, maintains a continuous trunk, selects branch nodes based on quality, exploration, and complementarity, and adapts branch-update granularity based on whether the trunk update is accepted.

Trunk-Guided Branch Update. This asymmetry between the full-batch trunk and the partial-batch branches motivates a trunk-guided sampling scheme. At each training step, SkillVine employs a trunk-and-branch sampling scheme. It first updates the trunk incrementally with the current batch. The trunk's accept/reject decision then serves as an information-quality signal for that batch. If the trunk accepts the update, the batch is judged to include more useful information than harmful or malicious information. The node selector can therefore select a promising historical checkpoint and apply a more radical branch update using this batch. If the trunk rejects the update, the batch is judged to contain insufficient useful information or potentially harmful information. The corresponding branch update should therefore be made more carefully, e.g., through a more conservative update or a stricter acceptance criterion. The node selector identifies promising historical checkpoints for side-branch update, prioritizing versions that remain promising yet underexplored. After training, SkillVine selects the skill library version with the highest validation accuracy from all trunk and branch nodes in the graph as the final output.

3.3 SkillVine Methodology

Skill Library Version Graph Representation. To ensure full recoverability during evolution, SkillVine organizes all skill library versions into a version graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $v \in \mathcal{V}$ corresponds to a specific skill library version S_v, and each edge $(u, v) \in \mathcal{E}$ indicates that S_v is derived by updating its parent version S_u. This graph preserves every historical version as a recoverable checkpoint, enabling future updates to revert to any earlier node and branch off for alternative exploration paths.

Each node v in the graph stores three types of metadata. First, for position and role, a *parent pointer* $p(v)$ identifies its parent, and a *node state* $z(v)$ marks it as initial, accepted, rejected, current, or best. Second, for performance characterization, the node keeps a *validation accuracy* $q(v)$ alongside the explicit sets of *solved and failed validation samples* $(\mathcal{D}_v^+, \mathcal{D}_v^-)$, which collectively offer a more concrete and diagnostic view of its capabilities than a scalar accuracy alone. Third, for exploration tracking, the *expansion count* $n_{\text{exp}}(v)$ records how many child versions have already been generated from this node, serving as a key signal for the node selector to prioritize checkpoints that remain promising yet underexplored. Together with the saved skill content S_v, these metadata allow SkillVine to restore any historical version and explore new update paths from it.

Trunk-and-Branch Evolution Strategy. SkillVine avoids randomly selecting historical nodes for branching exploration. Instead, it maintains a trunk evolution path for continuous skill refinement while using side branches to explore skill capabilities that the trunk lacks. Training proceeds batch by batch: at each step, a batch of training tasks arrives, and SkillVine performs three operations: (1) update the trunk, (2) select historical nodes as branching candidates using the node selector, and (3) optimize each selected node, where the optimization granularity is determined by whether the trunk accepts its own update. The trunk and each branch collect their own trajectories on the same batch.

Through this evolution strategy, the trunk follows a continuous optimization path that observes every batch and accumulates long-horizon information, whereas branches explore alternative evolution paths that may use only a subset of the training batches.

Trunk Node Update. At training step t, let $u_t^{\mathrm{tr}} \in \mathcal{V}$ denote the current trunk parent node. Based on the agent interaction trajectories collected at this step, SkillVine updates u_t^{tr} and produces a candidate child node v_t^{tr} for the trunk. It then evaluates the candidate on the validation set and accepts or rejects the update based on the validation result. Specifically, as in SkillOpt-Lite (Shen et al., 2026), the trunk generates several edits $\mathcal{P}_t^{\mathrm{tr}} = (\xi_{t,1}^{\mathrm{tr}}, \ldots, \xi_{t,m_t}^{\mathrm{tr}})$ from the training trajectories at the current step and then applies all these edits to its parent node. The update is accepted only if the modified skill achieves higher validation accuracy than its parent; otherwise, it is rejected:

$$a_t^{\mathrm{tr}} = \mathbb{I}\left[q\left(v_t^{\mathrm{tr}}\right) > q\left(u_t^{\mathrm{tr}}\right)\right]. \tag{2}$$

Branch Node Selection. At each training step, after the trunk update, SkillVine performs k side-branch updates. For each branch i, if its update was accepted at the previous step, it continues from its latest accepted version. Otherwise, the node selector chooses a historical version as its new parent. At the first training step, all branch parents are chosen by the node selector. Let $u_t^{(i)}$ denote the parent of branch i at step t. Each branch updates its parent using the same training batch as the trunk. The node selector identifies historical versions with potential for further improvement. For node v, it computes a step-dependent score that combines three terms:

$$\mathrm{Select}_t(v) = q(v) + \lambda_{\mathrm{exp}} \sqrt{\frac{\log\left(N_{\mathrm{exp}} + 1\right)}{n_{\mathrm{exp}}(v) + 1}} + \lambda_{\mathrm{comp}} C_t(v). \tag{3}$$

Here, $q(v)$ is the validation accuracy of node v. The second term gives a larger bonus to nodes that have produced fewer children, where $n_{\mathrm{exp}}(v)$ is the number of candidate child versions generated and evaluated from node v, including both accepted and rejected versions, and $N_{\mathrm{exp}} = \sum_{u \in \mathcal{V}} n_{\mathrm{exp}}(u)$ is the total number of children generated from all nodes. The third term $C_t(v)$ compares node v with the current trunk parent u_t^{tr}. It gives credit when v solves a validation sample that u_t^{tr} fails, and applies a penalty when v fails a sample that u_t^{tr} solves:

$$C_t(v) = \frac{\left|\mathcal{D}_v^{+} \cap \mathcal{D}_{u_t^{\mathrm{tr}}}^{-}\right|}{\left|\mathcal{D}_{u_t^{\mathrm{tr}}}^{-}\right|} - \beta \frac{\left|\mathcal{D}_{u_t^{\mathrm{tr}}}^{+} \setminus \mathcal{D}_v^{+}\right|}{\left|\mathcal{D}_{u_t^{\mathrm{tr}}}^{+}\right|}. \tag{4}$$

The coefficients λ_{exp} and λ_{comp} control the weights of exploration and complementarity, while β penalizes losing samples solved by the current trunk. $C_t(v)$ is essential to our branching strategy. We select nodes based not only on their validation accuracy and expansion counts, but also on whether they solve cases missed by the trunk. This allows us to explore a wider range of skill evolution paths within a limited training budget. In SkillVine, the node selector coefficients can be set separately for each of the k side-branch updates at a training step. Larger λ_{exp} and λ_{comp} values favor underexplored, complementary nodes, while smaller values favor validation accuracy. Empirically, mixing selectors with different selection preferences yields more consistent gains.

Adaptive-Granularity Branch Update (AGBU). At each training step, after the trunk is updated and the branch nodes are selected, each branch collects trajectories on the same training batch and generates several skill edits. Some edits generated from the training batch improve validation performance, while others hurt it. Their combined effect determines whether the update is accepted. Since the trunk and branches use the same training data at each step, if this batch produces harmful edits for the trunk, it is also likely to produce harmful edits for the branches. To avoid similar harmful edits in the branches, we use the outcome of the trunk as a signal to adjust the granularity of branch updates. Each branch generates edits $\mathcal{P}_t^{(i)} = (\xi_{t,1}^{(i)}, \ldots, \xi_{t,m_t^{(i)}}^{(i)})$ from the current training batch.

If the trunk update is accepted, i.e., $a_t^{\mathrm{tr}} = 1$, the current training batch is likely to produce more beneficial edits. The branches therefore use a coarse-grained update. Each branch applies all its edits to its parent node at once. Let $v_t^{(i)}$ denote the child node produced by applying all edits to $u_t^{(i)}$:

$$S_{v_t^{(i)}} = \mathcal{A}\left(S_{u_t^{(i)}}, \mathcal{P}_t^{(i)}\right), \qquad \text{if } a_t^{\mathrm{tr}} = 1, \tag{5}$$

Algorithm 1 SkillVine

Require: Initial skill library S_0; training batches $\{\mathcal{D}_t\}_{t=1}^{T}$; validation set $\mathcal{D}_{\text{val}}$; branch budget k
Ensure: Final skill library $S^\star$

Initialize the skill library version graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with root v_0 and skill $S_{v_0} \leftarrow S_0$
Evaluate v_0 on $\mathcal{D}_{\text{val}}$ and set the trunk parent $u_1^{\text{tr}} \leftarrow v_0$
for $t = 1, \ldots, T$ **do**
    Generate trunk edits $\mathcal{P}_t^{\text{tr}}$ from trajectories on $\mathcal{D}_t$
    Apply all edits to $S_{u_t^{\text{tr}}}$ and evaluate the candidate v_t^{tr} on $\mathcal{D}_{val}$
    $u_{t+1}^{\text{tr}} \leftarrow v_t^{\text{tr}}$ if $q(v_t^{\text{tr}}) > q(u_t^{\text{tr}})$ (accept); otherwise $u_{t+1}^{\text{tr}} \leftarrow u_t^{\text{tr}}$ (reject)
    Record v_t^{tr} and its status in $\mathcal{G}$; update its parent link and the parent's expansion count
    for $i = 1, \ldots, k$ **do**
        if $t > 1$ and branch i produced an accepted update in training step $t-1$ **then**
            Set $u_t^{(i)}$ to its latest accepted node from step $t-1$
        else
            Set $u_t^{(i)}$ to the next parent ranked by equation 3, using u_t^{tr} as the reference
    for all $i = 1, \ldots, k$ **in parallel do**
        Collect trajectories using $S_{u_t^{(i)}}$ on $\mathcal{D}_t$ and generate edits $\mathcal{P}_t^{(i)}$
        if the trunk update was accepted **then**
            Apply $\mathcal{P}_t^{(i)}$ at once using equation 5 and evaluate $v_t^{(i)}$ on $\mathcal{D}_{val}$
            Accept if $q(v_t^{(i)}) > q(u_t^{(i)})$; otherwise reject
        else
            Set the initial edit parent $u_{t,1}^{(i)} \leftarrow u_t^{(i)}$
            for each edit $\xi_{t,j}^{(i)}$ in order **do**
                Apply $\xi_{t,j}^{(i)}$ to $S_{u_{t,j}^{(i)}}$ using equation 6 and evaluate $v_{t,j}^{(i)}$ on $\mathcal{D}_{val}$
                Accept if $q(v_{t,j}^{(i)}) > q(u_{t,j}^{(i)})$; otherwise reject
                Set $u_{t,j+1}^{(i)} \leftarrow v_{t,j}^{(i)}$ if accepted; otherwise set $u_{t,j+1}^{(i)} \leftarrow u_{t,j}^{(i)}$
    Record all branch nodes and their statuses in $\mathcal{G}$; update parent links and expansion counts
Select the highest-scoring initial or accepted node $v^\star$ in $\mathcal{G}$
return $S_{v^\star}$

where $\mathcal{A}$ denotes applying edits to a skill. The update is accepted if validation accuracy improves.

If the trunk update is rejected, i.e., $a_t^{\text{tr}} = 0$, the training batch is more likely to produce harmful edits. The branches then use a fine-grained update. Each branch applies its edits one at a time and in order. After each edit, the resulting skill is evaluated on the validation set. If the edit is accepted, the resulting version becomes the parent for the next edit. Otherwise, the branch returns to the last accepted version. Let $u_{t,j}^{(i)}$ denote the latest accepted node before applying the j-th edit and $v_{t,j}^{(i)}$ denote the candidate child node produced by the j-th edit , where $u_{t,1}^{(i)} = u_t^{(i)}$:

$$S_{v_{t,j}^{(i)}} = \mathcal{A}\left(S_{u_{t,j}^{(i)}}, \xi_{t,j}^{(i)}\right), \qquad \text{if } a_t^{\text{tr}} = 0. \tag{6}$$

All resulting nodes are recorded in the version graph. A coarse-grained update requires only one evaluation and is therefore less costly. A fine-grained update requires multiple evaluations, but it reveals the effect of each edit separately and helps filter out harmful edits. These two update strategies involve a trade-off between evaluation cost and update reliability. We use the outcome of the trunk update as a signal to adapt the branch update strategy at each training step.

Best Skill Output. After training, SkillVine returns $S^\star = S_{v^\star}$, where $v^\star = \arg\max_{v \in \mathcal{V}} q(v)$. Algorithm 1 summarizes the procedure.

4 EXPERIMENTS

In this section, we evaluate SkillVine against six baselines, assess its components through ablations, and analyze evolution dynamics and system parameters.

Table 1: Main results across benchmarks. Higher values indicate better performance.

Model	Method	Benchmark Score				
		Spreadsheet	LiveMath	SearchQA	OfficeQA	TABVERSE
DeepSeek-V4 -flash	Linear	48.04	43.40	73.36	47.30	63.88
	Best-of-3	64.41	50.00	73.14	45.27	64.49
	Random	53.38	46.23	74.14	52.70	64.08
	SkillOpt	62.99	47.17	73.50	46.62	63.47
	SkillOpt-Lite	66.19	50.48	73.14	37.84	61.84
	Hermes-SE	50.53	51.89	73.57	53.38	60.82
	SkillVine	**71.53**	**53.77**	**74.43**	**55.41**	**67.35**
Qwen3.8 -Flash	Linear	66.90	36.79	72.14	50.00	60.82
	Best-of-3	65.12	32.08	72.21	47.30	60.82
	Random	66.90	37.74	72.14	50.00	61.22
	SkillOpt	65.12	36.79	72.21	43.24	61.63
	SkillOpt-Lite	65.84	33.02	71.93	51.91	59.39
	Hermes-SE	66.19	**42.45**	71.86	47.97	62.24
	SkillVine	**67.97**	39.62	**72.40**	**54.05**	**63.67**

4.1 Experimental Setup

Benchmarks. We evaluate SkillVine on five complementary benchmarks: **SpreadsheetBench** (Ma et al., 2024) for instruction-based Excel editing; **LiveMath** (He et al., 2026) for multiple-choice mathematical reasoning; **OfficeQA** (Opsahl-Ong et al., 2026) for question answering over office and government documents; **SearchQA** (Dunn et al., 2017) for open-domain answering with noisy retrieved passages; and **TABVERSE-QA** (Ahsan et al., 2026) for table-based lookup, comparison, calculation, and verification.

Baselines. We compare SkillVine with six baselines: **Linear:** Skills evolve along a single linear path. **Best-of-3**: This baseline maintains a linear evolution path but generates and evaluates three candidate updates per step, selecting the one with the highest validation score. **Random**: It uses the same number of branch expansion attempts as SkillVine but randomly selects parent nodes. **Hermes-SE** (Nous Research, 2026): It uses GEPA (Agrawal et al., 2026) to analyze execution traces, generate improved skill variants, and select better variants through evolutionary search. **SkillOpt:** (Yang et al., 2026a) It evolves the skill library linearly, compares previous and current skills after each epoch to add protected long-term guidance, and retains optimizer-side experience to guide subsequent edits. **SkillOpt-Lite:** (Shen et al., 2026) Built on SkillOpt, it replaces the fixed structured-edit pipeline with a coding agent that directly edits the skill file.

Implementation Details. We implement SkillVine on top of SkillOpt-Lite (Shen et al., 2026). We use DeepSeek-V4-flash (Xu et al., 2026) provided by Volcano Engine (Volcano Engine) and Qwen3.8-Flash (Alibaba Cloud, 2026) provided by Alibaba Cloud Model Studio (Alibaba Cloud, n.d.) as base models. We disable thinking mode for Qwen3.8-Flash. We follow the official SkillOpt-Lite repository's epoch, batch-size, and training-step settings. Unless otherwise specified, all methods with branch exploration use two branch update attempts per training step. For SkillVine, we use a mix of node selectors that emphasize exploration and complementarity and those that emphasize higher validation accuracy. We refer to them as **discovery selectors** and **refinement selectors**, respectively. We split the k side-branch update attempts per training step equally between the two, giving the discovery selectors one extra attempt when k is odd. We set $\lambda_{\text{exp}} = 0.08$ and $\lambda_{\text{comp}} = 0.5$ for discovery selectors, and set both to zero for refinement selectors. We set $\beta = 1.6$.

4.2 Main Results

Table 1 presents the test performance of SkillVine and 6 baselines across 5 benchmarks and 2 base LLMs. To reduce the effect of randomness in agent rollouts and save API calls, we first run Linear for each benchmark-model combination. Best-of-3, Random, and SkillVine reuse Linear's trunk results. Nodes from the trunk become available only when the corresponding step is reached, so no method can access results from future steps. Note that Best-of-3 reuses the Linear trunk node only when neither additional attempt achieves a higher validation score. Otherwise, it generates subsequent results independently. SkillOpt and SkillOpt-Lite reuse Linear's first-epoch results. These

Table 2: Ablation study results. Higher values indicate better performance.

Model	Ablation	Benchmark Score				
		Spreadsheet	LiveMath	SearchQA	OfficeQA	TABVERSE
DeepSeek-V4 -flash	w/o Selector & AGBU	53.38	46.23	74.14	52.70	64.08
	w/o Selector	67.26	46.23	72.93	47.30	63.47
	w/o AGBU	65.84	50.00	**74.57**	52.03	63.27
	w/o Comp	**71.53**	**53.77**	73.36	**55.41**	60.82
	SkillVine	**71.53**	**53.77**	74.43	**55.41**	**67.35**
Qwen3.8 -Flash	w/o Selector & AGBU	66.90	37.74	72.14	50.00	61.22
	w/o Selector	65.84	35.85	72.14	50.00	63.27
	w/o AGBU	65.48	37.74	**72.86**	51.35	62.65
	w/o Comp	66.55	**39.62**	71.07	**54.05**	61.43
	SkillVine	**67.97**	**39.62**	72.40	**54.05**	**63.67**

reuse does not change the methods. Table 1 shows that SkillVine achieves the highest test score on all the five benchmarks with DeepSeek-V4-flash and four of the five benchmarks with Qwen3.8-Flash. SkillVine achieves large gains over Linear, especially on SpreadsheetBench with DeepSeek-V4-flash, where the test score rises from 48.04 to 71.53, an increase of 23.49. This indicates the limitations of linear evolution and shows that branching exploration can find better skill library versions that linear evolution misses. SkillVine outperforms Best-of-3 in all 10 benchmark–model combinations. This suggests that SkillVine is more effective than simply increasing the number of update attempts. SkillVine also outperforms Random in all 10 benchmark–model combinations. This shows that SkillVine's guided branch exploration is more effective than branching from randomly selected nodes. The gains vary across benchmarks and base LLMs. Gains are larger on OfficeQA and SpreadsheetBench with DeepSeek-V4-flash, where errors are concentrated in a few types and the weaker trunk leaves more room for improvement. Gains are smaller on SearchQA, where errors are more diverse and the trunk has achieved high performance. SkillVine achieves the highest scores among all methods in most settings, supporting its broad applicability. The validation scores of the final best nodes are reported in Table 6 in the appendix.

4.3 Ablation Study

Table 2 presents the ablation results for SkillVine. For the w/o Selector setting, we replace our node selector with random selection while retaining AGBU. For both base LLMs, SkillVine outperforms most ablation variants with components removed. On SearchQA, as the test scores are close and agent rollouts are stochastic, SkillVine has a slightly lower test score than w/o AGBU. However, the validation score achieved by the final best node of SkillVine is still the highest. Removing complementarity changes parent-node selection in five of the ten model–benchmark combinations. To reduce rollout variance, we fully reuse SkillVine's training results for the five unchanged combinations; for the others, we reuse results up to the first selection divergence and rerun from that step onward with the complementarity term removed. Table 2 suggests that complementarity helps discover better alternative paths. Figure 2 compares training cost, measured by the number of agent rollouts, and performance across the five settings. Branch exploration and AGBU require additional rollouts, but both improve performance. The w/o Selector setting uses slightly more rollouts than SkillVine but performs worse, showing that our selector uses the branch budget more effectively.

4.4 Analysis

4.4.1 Skill Evolution Dynamics

Figure 3 shows the best validation score found up to each training step on SpreadsheetBench. SkillVine achieves the highest score at every step across all eight training steps, reaching 76.92%. The final best node achieves the highest test score of 71.53%. We further examine whether the best nodes come from the trunk or branches. Figure 4 shows the percentage of training steps in which the global-best node comes from each source. Branches provide the global-best node in a large proportion of training steps. On SpreadsheetBench and OfficeQA, the global-best node even come from branch exploration at every training step. A closer analysis shows that, across all ten benchmark–model combinations, the final best nodes are reached through branch paths. These results indicate

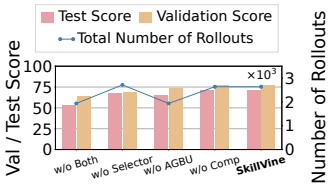


Figure 2: Training cost and performance on SpreadsheetBench with two branch update attempts per training step.

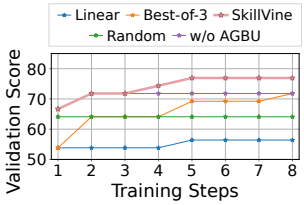


Figure 3: Validation score of the best node found up to each training step on SpreadsheetBench.

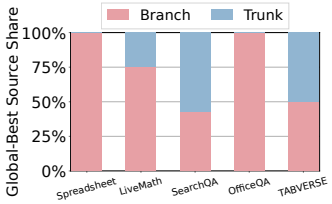
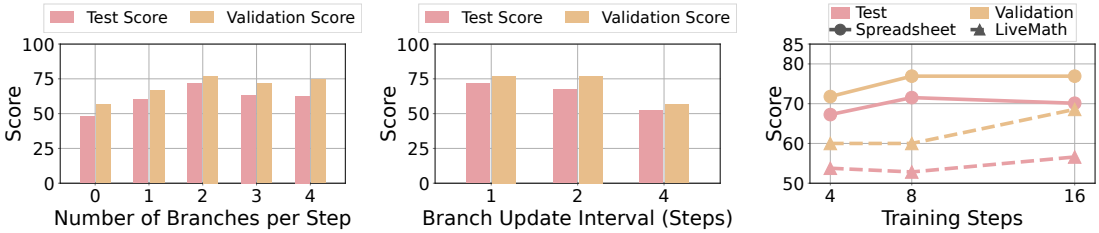


Figure 4: Proportion of training steps in which the best node comes from the trunk or side branches.



(a) Impact of branches numbers. (b) Impact of branch update interval. (c) Impact of training steps.

Figure 5: Impact of system parameters on validation and test scores with DeepSeek-V4-flash. Panels (a) and (b) use SpreadsheetBench; panel (c) includes both SpreadsheetBench and LiveMath.

that branch exploration helps find better skill library versions than those available along the trunk. Among the ten paths, two paths use only discovery selectors, two only refinement selectors, and six both, demonstrating that both selector types contribute to finding the best paths. AGBU rejects at least one harmful edit in eight paths, helping preserve promising evolution paths. Table 3 details all ten paths.

4.4.2 Impact of System Parameters

Figure 5 shows how three system parameters affect validation and test scores: the number of branch update attempts per training step, the branch update interval, and the number of training steps. Figure 5a shows that performance generally improves at first and then levels off as the number of branch update attempts per training step increases. Even one attempt improves performance, while two attempts already provide strong gains. Figure 5b shows that shorter branch update intervals lead to better performance. A shorter interval allows branch updates at more training steps and produces more candidate nodes, increasing the chance of finding a high-performing branch node. Figure 5c shows that using smaller batches and more training steps generally improves performance. With a fixed number of branch update attempts per step, more steps provide more rounds of branch exploration and produce more candidate nodes, increasing the chance of finding a high-performing node. More steps also divide evolution into more stages, allowing updates to be added and evaluated gradually and reducing the risk of harmful edits accumulating in a single update.

5 Conclusion

In this paper, we introduce SkillVine, an automatic skill-evolution framework that enables parallel search through branching exploration. Its core components include a trunk-branch collaborative search mechanism, a complementarity-aware node selection method, and an adaptive-granularity update rule. Overall, SkillVine outperforms six baselines in nine of ten model–benchmark combinations and finds better skill-library versions through branching than along the linear trunk, offering a new approach to agent self-evolution.

AI Use Statement

We used generative AI tools to assist with writing and language polishing, literature retrieval and discovery, research ideation and execution, and drafting sections of the paper. We did not use generative

AI tools to generate synthetic datasets or prove mathematical claims. We reviewed all AI-assisted work and take responsibility for the final content of this paper, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Momina Ahsan, Sarfraz Ahmad, Ming Shan Hee, Roy Ka-Wei Lee, and Preslav Nakov. TABVERSE: Benchmarking cross-format table understanding in LLMs and VLMs, 2026. URL https://arxiv.org/abs/2606.09578.

Alibaba Cloud. qwen3.8-flash. https://www.alibabacloud.com/help/en/model-studio/qwen3-8-flash, 2026. Accessed: September 9, 2026.

Alibaba Cloud. Alibaba Cloud Model Studio. https://www.alibabacloud.com/en/product/modelstudio, n.d. Accessed: September 9, 2026.

Adrian Bolton, Alexander Lerchner, Alexandra Cordell, Alexandre Moufarek, Andrew Bolt, Andrew Lampinen, Anna Mitenkova, Arne Olav Hallingstad, Bojan Vujatovic, Bonnie Li, et al. Sima 2: A generalist embodied agent for virtual worlds. *arXiv preprint arXiv:2512.04797*, 2025.

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.

Yuxuan Cai, Yipeng Hao, Jie Zhou, Hang Yan, Zhikai Lei, Rui Zhen, Zhenhua Han, Yutao Yang, Junsong Li, Qianjun Pan, Tianyu Huai, Qin Chen, Xin Li, Kai Chen, Bo Zhang, Xipeng Qiu, and Liang He. Building self-evolving agents via experience-driven lifelong learning: A framework and benchmark, 2026. URL https://arxiv.org/abs/2508.19005.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, volume 2025, pp. 50466–50494, 2025.

Matthew Dunn, Levent Sagun, Mike Higgins, V Ugur Guney, Volkan Cirik, and Kyunghyun Cho. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*, 2017.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

EvolvingLMMs-Lab. SkillOpt-Lite: Agentic skill optimization via reflective training loops. https://github.com/EvolvingLMMs-Lab/SkillOpt-Lite, 2026. GitHub repository, accessed August 19, 2026.

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

Google DeepMind. Introducing the Gemini 2.5 Computer Use model. https://blog.google/innovation-and-ai/models-and-research/google-deepmind/gemini-computer-use-model/, October 2025. Accessed: 2026-06-25.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. Rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.

Linyang He, Qiyao Yu, Hanze Dong, Baohao Liao, Xinxing Xu, Micah Goldblum, Jiang Bian, and Nima Mesgarani. Livemathematicianbench: A live benchmark for mathematician-level reasoning with proof sketches. *arXiv preprint arXiv:2604.01754*, 2026.

Zisu Huang, Jingwen Xu, Yifan Yang, Ziyang Gong, Qihao Yang, Muzhao Tian, Xiaohua Wang, Changze Lv, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Xue Yang, Dongdong Chen, Xiaoqing Zheng, and Chong Luo. From raw experience to skill consumption: A systematic study of model-generated agent skills, 2026. URL https://arxiv.org/abs/2605.23899.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024.

Yubo Li. Dynamic agent skills: A lifecycle survey and taxonomy of evolving skill libraries. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=cjU3YbcRr8. Survey Certification.

Zujie Liang, Feng Wei, Wujiang Xu, Yuxi Qian, Lin Chen, and Xinhui Wu. I-mcts: Enhancing agentic automl via introspective monte carlo tree search. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 189–210, 2026.

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.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

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. *Advances in Neural Information Processing Systems*, 37:94871–94908, 2024.

Bo Mao, Jie Zhou, Yutao Yang, Xin Li, Xian Wei, Qin Chen, Xingjiao Wu, and Liang He. Learning while acting: A skill-enhanced test-time co-evolution framework for online lifelong learning agents, 2026. URL https://arxiv.org/abs/2606.04815.

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.

Nous Research. Hermes agent self-evolution. https://github.com/NousResearch/hermes-agent-self-evolution, 2026. Software repository, accessed September 7, 2026.

OpenAI. Computer-Using Agent. https://openai.com/index/computer-using-agent/, 2025a. Accessed: 2026-06-25.

OpenAI. Introducing deep research. https://openai.com/index/introducing-deep-research/, February 2025b. Accessed: 2026-06-25.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, et al. Officeqa pro: An enterprise benchmark for end-to-end grounded reasoning. *arXiv preprint arXiv:2603.08655*, 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*, 2026.

Qianjun Pan, Yutao Yang, Junsong Li, Jie Zhou, Kai Chen, Xin Li, Qin Chen, and Liang He. Anything2skill: Compiling external knowledge into reusable skills for agents, 2026. URL https://arxiv.org/abs/2606.09316.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Yifei Shen, Bo Li, and Xinjie Zhang. Skillopt-lite: Better and faster agent self-evolution via one line of vibe. *arXiv preprint arXiv:2607.03451*, 2026.

Hongwen Song and Song Wei. More skills, worse agents? skill shadowing degrades performance when expanding skill libraries, 2026. URL https://arxiv.org/abs/2605.24050.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, et al. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*, 2025.

Volcano Engine. Volcano engine. https://www.volcengine.com/. Accessed: August 19, 2026.

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.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, et al. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.

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.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

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, Bo Zhang, and Liang He. Autoskill: Experience-driven lifelong learning via skill self-evolution, 2026b. URL https://arxiv.org/abs/2603.01145.

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, 2022.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with agent skills. Anthropic Engineering Blog, October 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills. Accessed: 2026-06-25.

Qiang Zhang, Ruixue Ding, Fanrui Zhang, Xi Chen, Boli Chen, Shihang Wang, Yinfeng Huang, Yi Zheng, Pengjun Xie, Kaipeng Zhang, Jiawei Liu, and Zheng-Jun Zha. ArenaFlow: From trajectory ranking to hierarchical credit propagation for open-ended agent RL. *arXiv preprint arXiv:2609.21378*, 2026. URL https://arxiv.org/abs/2609.21378.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

A Appendix

A.1 SkillVine Evolution Paths across 10 Benchmark–Model Combinations

Table 3: Evolution paths to the final best nodes across the ten benchmark–model combinations. Each vXXXX is a skill library version number within that run. **M:vXXXX** denotes a node on the trunk; **D:vXXXX** denotes a side-branch node generated using a discovery selector; **R:vXXXX** denotes a side-branch node generated using a refinement selector. † means that AGBU rejects at least one edit during the update.

Model	Benchmark	Evolution Path
DeepSeek-V4 -flash	SpreadsheetBench	M:v0001 → D:v0004 → R:v0007† → D:v0026† → D:v0030
	LiveMath	M:v0001 → D:v0004 → R:v0006 → D:v0011†
	SearchQA	M:v0001 → D:v0004 → D:v0010 → D:v0041† → R:v0187†
	OfficeQA	M:v0001 → R:v0009
	TABVERSE	M:v0001 → D:v0004 → D:v0010†
Qwen3.8 -Flash	SpreadsheetBench	M:v0001 → R:v0003 → R:v0013† → D:v0027†
	LiveMath	M:v0001 → R:v0003
	SearchQA	M:v0001 → M:v0002 → M:v0005 → M:v0017 → M:v0020 → M:v0113 → D:v0227†
	OfficeQA	M:v0001 → D:v0004 → R:v0010 → R:v0012† → R:v0018† → R:v0024†
	TABVERSE	M:v0001 → D:v0007† → D:v0017† → R:v0029† → R:v0037† → D:v0065†

Table 3 presents the evolution paths leading to the final best nodes across the ten benchmark–model combinations. Each path traces the sequence of skill library version numbers from the initial node to the final best node. As shown in Table 3, all ten best nodes are reached through paths that diverge from the trunk, showing that better paths are common but often missed by linear evolution, whereas SkillVine can discover these better paths through branch exploration. Among these ten paths, two paths contain only discovery selector updates, two paths contain only refinement selector updates, and the remaining six paths contain both, showing that both selectors contribute to finding paths to the best nodes. In eight of these ten paths, AGBU rejects at least one harmful edit, showing that it helps prevent harmful edits from disrupting promising paths to the best nodes.

A.2 Examples of Complementary Nodes

Table 4 presents two examples of complementary nodes selected by SkillVine. In Case 1, R:v0007 solves seven validation samples missed by M:v0002 while retaining all 21 samples solved by the trunk. In Case 2, D:v0010 solves nine samples missed by M:v0031 but loses five samples solved by the trunk. In both cases, evolution from the selected branch produces a final node that outperforms the best trunk node on both validation and test sets. These examples show that the complementarity-aware selector preserves useful alternative capabilities and opens better evolution paths.

A.3 Examples of Harmful Edits Rejected by AGBU

Table 5 presents two examples of how AGBU filters harmful edits from rejected trunk-update batches. In Case 1, the trunk update introduces an overgeneralized rule that always returns the full cell text. This rule loses more validation samples than it gains, causing the trunk update to be rejected. During the branch update, AGBU rejects the same harmful edit but retains a useful edit

Table 4: Two examples of selected complementary nodes.

Case 1: DeepSeek-V4-flash × SpreadsheetBench

Version Node	Skill Capability	Unique / Lost	Typical Validation Samples	Final Gain
Trunk **M:v0002** Branch **R:v0007**	**R:v0007 adds Python-based calculation beyond M:v0002's workbook inspection and preservation.** **M:v0002** rules: Inspect headers and cell types; read cached formula values while preserving the workbook. **R:v0007** rules: Compute in Python and write values directly; infer the intended operation from the actual table when the supplied formula is flawed.	**M:v0002** 0 / 7 **R:v0007** 7 / 0	**10747: Two-key profit lookup** **M:v0002**: `None` (fail) **R:v0007**: `-10600` (pass) **51556: Conditional average** **M:v0002**: `None` (fail) **R:v0007**: `82` (pass) The branch also solves all 21 examples solved by the trunk.	Final best **D:v0030** outperforms the best trunk **M:v0028** by 20.51 on validation and 23.49 on test.

Case 2: DeepSeek-V4-flash × SearchQA

Version Node	Skill Capability	Unique / Lost	Typical Validation Samples	Final Gain
Trunk **M:v0031** Branch **D:v0010**	**D:v0010 handles wordplay and answer forms missed by M:v0031; M:v0031 better resolves associated entities in the examples shown.** **M:v0031** rule: For a bare entity name, identify the associated category or relationship from context. **D:v0010** rules: Match the source answer form; for word transformations, return both the original and transformed terms.	**M:v0031** 5 / 9 **D:v0010** 9 / 5	**"Adam Levine"** **M:v0031**: Maroon 5 (pass) **D:v0010**: Adam Levine (fail) **Holes on postage stamps** **M:v0031**: perforation (fail) **D:v0010**: perforations (pass) **Hawaiian wreath becoming a sheltered area** **M:v0031**: lee (fail) **D:v0010**: lei & lee (pass)	Final best **R:v0187** outperforms the best trunk **M:v0134** by 5.00 on validation and 1.07 on test.

for selecting the correct column and computing durations, allowing R:v0007 to improve from 47/70 to 49/70 and be accepted. In Case 2, the trunk update introduces an overly broad name-shortening rule that reduces the validation score from 152/200 to 148/200. AGBU rejects the same rule on the branch while retaining two useful edits for selecting the answer type and formatting paired answers, allowing D:v0066 to improve from 147/200 to 151/200 and be accepted. These cases show that AGBU can separate harmful edits from useful ones and preserve the beneficial parts of an otherwise rejected update.

Table 5: Two examples of harmful edits rejected by AGBU.

Version Node	Rejected Trunk Update	AGBU-Filtered Branch Update	Accepted Gain
Case 1: DeepSeek-V4-flash × TABVERSE			
Mainline **M:v0002** → **M:v0005** (rejected) Branch **D:v0004** → **R:v0007**†	**Proposed rule:** Return the full cell text, such as "Middelkerke, Belgium", rather than one part. **Result:** The update included this rule. It lost three previously solved samples and only gained two: 45/70 → 44/70 (−1.43), so it was rejected.	AGBU **kept one edit and rejected two**. The kept edit selects the correct column and computes durations from date ranges: 47/70 → 49/70. An edit with the same full-cell rule then reduced the score to 43/70 (−8.57), so AGBU rejected it.	The accepted child **R:v0007**† improved over **D:v0004**: 47/70 → 49/70, **+2 samples** (**+2.86**).
Case 2: Qwen3.8-Flash × SearchQA			
Mainline **M:v0020** → **M:v0058** (rejected) Branch **M:v0002** → **D:v0066**†	**Proposed rule:** Shorten names: use "Webster" instead of "Noah Webster" and "the Colorado" instead of "Colorado River". **Result:** The update included this rule. The score fell from 152/200 to 148/200 (−2.00), so it was rejected.	AGBU **kept two edits and rejected two**. An edit with the same name-shortening rule reduced the score from 147/200 to 141/200 (−3.00), so it was rejected. The kept edits select the specific answer type and use "&" for paired answers such as "Nina & Pinta". Together, they raised the score to 151/200.	The accepted child **D:v0066**† improved over **M:v0002**: 147/200 → 151/200, **+4 samples** (**+2.00**).

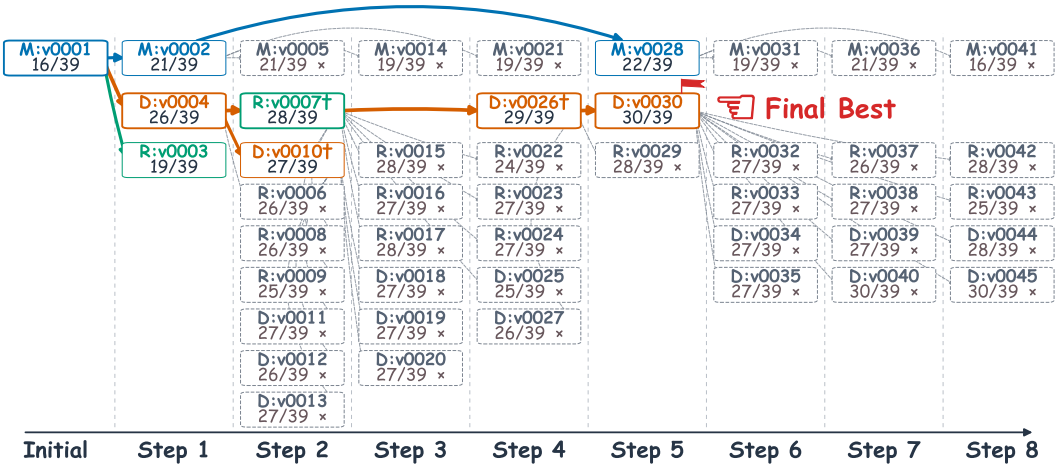


Figure 6: Evolution path of DeepSeek-V4-flash on SpreadsheetBench. Nodes are labeled with their version numbers and validation scores. Gray dashed nodes denote rejected candidate versions. Blue arrows denote accepted trunk updates, while orange and green arrows denote accepted side-branch updates using the discovery and refinement selectors, respectively. † marks updates for which AGBU rejects at least one edit.

A.4 An Example of Evolution Process

Figure 6 shows an example of the evolution process. The trunk first improves from 16/39 to 21/39, but most of its later updates are rejected, and its best score is only 22/39. In contrast, the branch path continues to improve and reaches the final best node D:v0030 with a score of 30/39. This path uses both discovery and refinement selectors. At R:v0007 and D:v0026, AGBU filters out harmful edits while keeping useful ones. This example shows how branch exploration finds a better path when linear evolution stops improving.

Table 6: Validation results across benchmarks. Higher values indicate better performance.

Model	Method	Validation Score				
		Spreadsheet	**LiveMath**	**SearchQA**	**OfficeQA**	**TABVERSE**
DeepSeek-V4 -flash	Linear	56.41	54.29	77.00	61.22	70.00
	Best-of-3	71.79	57.14	77.00	65.31	**71.43**
	Random	64.10	54.29	77.00	67.35	70.00
	SkillOpt	71.79	**60.00**	78.00	57.14	70.00
	SkillOpt-Lite	71.79	45.71	76.00	53.06	68.57
	Hermes-SE	61.54	57.14	76.50	69.39	67.14
	SkillVine	**76.92**	**60.00**	**82.00**	**71.43**	**71.43**
Qwen3.8 -Flash	Linear	69.23	37.14	77.50	67.35	71.43
	Best-of-3	**71.79**	40.00	77.50	69.39	71.43
	Random	69.23	48.57	77.50	67.35	71.43
	SkillOpt	69.23	**54.29**	**78.00**	59.18	67.14
	SkillOpt-Lite	66.67	34.29	77.00	69.39	68.57
	Hermes-SE	61.54	51.43	77.00	69.39	67.14
	SkillVine	**71.79**	48.57	**78.00**	**73.47**	**72.86**

A.5 Validation Results

Table 6 reports the validation scores of the best nodes used in Table 1. SkillVine achieves the highest validation score, including ties, in nine of the ten model–benchmark combinations. The only exception is LiveMath with Qwen3.8-Flash. This is because the validation set of LiveMath is small and agent rollouts are stochastic, making the final score more sensitive to individual samples. Different benchmark difficulties and data characteristics leave different amounts of room for improvement. For example, SkillVine shows larger gains on Spreadsheet and OfficeQA and smaller gains on SearchQA. Nevertheless, SkillVine consistently finds nodes with the highest validation scores across these tasks, demonstrating its broad applicability.