SKILL-FOREST: RANDOMIZED ENSEMBLES OF TEXTUAL SKILL OPTIMIZERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent Skills adapt frozen large language model agents by encoding reusable procedural knowledge outside model parameters. Under limited data, noisy textual updates and repeated selection on small validation sets can make single-path optimization sensitive to early decisions and evaluation noise.

We introduce Skill-Forest, an optimizer-level ensemble inspired by the randomize-and-aggregate principle of Random Forests. Each tree uses an independently sampled training–validation split and independently searches for its tree-local best Skill. Gain-prioritized expansion under a shared budget, validation-guided pruning, and early rejection enable efficient forest construction. The resulting tree-local best Skills are then aggregated into one deployable Skill through direct selection (Select) or hierarchical merge (Merge). This design diversifies both optimization paths and validation views to reduce the high variance associated with textual-gradient optimization.

We evaluate Skill-Forest across five benchmarks spanning direct-chat tasks, tool use, and embodied tasks. Skill-Forest Select and Merge improve the mean held-out Hard score over No-skill by 12.82 and 12.87 percentage points (pp), respectively. Under the same reflection strategy, they outperform SkillOpt by 2.59 and 2.63 pp, respectively. Skill-Forest reduces training and validation rollouts during forest construction by 73.5% across the five benchmarks relative to complete tree construction, supporting efficient multi-path ensemble Skill exploration.

1 INTRODUCTION

Large language model (LLM) agents have made rapid progress in reasoning, tool use, and code execution (Yao et al., 2023; Yang et al., 2024b; Li, 2025), yet reliable execution in specialized domains still requires domain-specific knowledge and procedures. Agent Skills provide a lightweight inference-time adaptation that externalizes this knowledge without updating model parameters (Xu & Yan, 2026; Li et al., 2026). However, manually authored or one-shot generated Skills can be incomplete or mismatched to the target task distribution. Recent methods therefore refine Skills iteratively by learning from execution trajectories, proposing textual revisions, and evaluating candidate Skills (Yang et al., 2026; Wang et al., 2026; Zheng et al., 2026).

Despite this progress, single-path optimization remains vulnerable to **noisy, path-dependent updates**. LLM-generated textual feedback is a sampled diagnosis rather than an objective-derived numerical gradient; the same execution evidence can produce different or even harmful revisions (Yuksekgonul et al., 2025; Yang et al., 2026). Each accepted revision also changes the trajectories available for subsequent learning, allowing early decisions to shape the remaining optimization path. A validation gate can reject an unhelpful candidate, but validation itself is myopic, assessing immediate performance rather than the longer-term value of an update path. Consequently, a promising direction may be discarded before its longer-term value becomes apparent.

A second challenge is **optimization and selection under limited data**. Scarce execution evidence must support both Skill revision and candidate evaluation, while evaluation on a small validation set is itself subject to substantial variability. Repeatedly selecting against one small validation set can favor rules specific to that validation view or accept ineffective updates because of evaluation noise (Yang et al., 2026; Peng et al., 2026). These limitations are particularly consequential when a less capable model must both execute tasks and diagnose its own failures, as such models often struggle

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.

to identify the underlying causes of errors accurately. Stronger teacher or optimizer models can help (Shi et al., 2026; Yang et al., 2026), but leave open how to improve Skills efficiently without a stronger optimizer.

We propose **Skill-Forest** for a **capability-matched** setting in which the same less capable backbone both executes tasks and optimizes its Skills using only outcome-level supervision. Skill-Forest is an optimizer-level ensemble inspired by the randomize-and-aggregate principle of Random Forests (Breiman, 2001). Each tree evolves from the same initial Skill on an independently sampled training–validation split, while branching preserves alternative update paths within each tree. A shared expansion budget prioritizes promising candidates and prunes weak ones early. Finally, tree winners are selected or merged into a single deployable Skill. This design enables broader and more diverse Skill exploration without relying on a stronger optimizer.

Our main contributions are:

- We introduce **Skill-Forest**, an optimizer-level ensemble formulation for capability-matched Skill optimization. It combines multiple independent trees, persistent branching, and tree-specific training–validation splits to diversify both optimization paths and validation views.
- We develop an efficient forest-construction strategy combining a forest-wide expansion budget, validation-gated pruning, gain-prioritized node expansion, and early rejection. These mechanisms focus computation on promising branches and avoid unnecessary validation rollouts.
- We validate Skill-Forest across five benchmarks spanning single-round direct-chat tasks, multi-round tool use, and persistent environment interaction. The results demonstrate the aggregate performance advantages and validation-rollout efficiency of this new ensemble-based Skill-learning paradigm, without a stronger optimizer model.

2 Related Work

2.1 Textual Optimization and Candidate Search

Optimizing text from execution feedback predates Skill optimization. ProTeGi (Pryzant et al., 2023) and OPRO (Yang et al., 2024a) iteratively generate and select textual candidates. TextGrad (Yuksekgonul et al., 2025) uses language-model feedback as textual gradients. GEPA (Agrawal et al., 2026) combines trajectory reflection with Pareto-based candidate selection and optionally merges candidates with complementary module updates.

2.2 Experience-Driven Skill Evolution

Execution experience can improve frozen agents through persistent external knowledge. Reflexion (Shinn et al., 2023) stores verbal feedback in episodic memory, Voyager (Wang et al., 2024) accumulates executable skills, and ACE (Zhang et al., 2026b) maintains an evolving playbook. More recent methods directly evolve textual Skills from execution. Trace2Skill (Ni et al., 2026) consolidates trajectory-level lessons; SkillCAT (Chen et al., 2026) derives and validates patches from comparative executions; SkillCommit (He & Yang, 2026) generalizes instance-specific patches through cross-instance validation; and WikiSkill (Tang et al., 2026) accumulates knowledge across evolution iterations.

Iterative Skill optimizers also refine how updates are generated and selected. SkillOpt (Yang et al., 2026) performs batch-wise updates with learning-rate constraints, SkillGrad (Wang et al., 2026) introduces textual diagnoses and momentum, and SkillProx (Zheng et al., 2026) incorporates utility-aware refinement. SSO (Peng et al., 2026) exploits comparative evidence from unlabeled instances.

2.3 Population Search and Optimizer Ensembles

Maintaining multiple candidates is an established strategy for automated agent design and program optimization. EvoSkill (Alzubi et al., 2026) retains a frontier of agent programs, SkillMOO (Gong et al., 2026) applies Pareto selection to Skill bundles, and CoEvoSkills (Zhang et al., 2026a) jointly evolves skills and verification. AFlow (Zhang et al., 2025) and OpenEvolve (Sharma, 2025) search

over agent workflows or executable programs, while Meta Context Engineering (MCE) (Ye et al., 2026) evolves the context-engineering procedure itself.

How Skill-Forest Differs. The studies discussed above provide methods for textual updates, Skill evolution, and candidate search. Skill-Forest runs multiple independent, tree-structured Skill optimizers, each with its own update state and sampled training–validation split. It then selects or merges the resulting tree winners into a single deployable Skill. This organization diversifies both optimization paths and validation views.

3 Problem Formulation

We formalize a general execution-grounded Skill optimization process. Given an initial Skill S_0, we consider a training task set $D = \{x_i\}_{i=1}^{N}$ sampled from a target task distribution $\mathcal{P}$. Each instance x_i consists of a task and its corresponding environment, which provides a reward signal based on the agent's performance. Our goal is to learn an improved Skill that enables an agent to generalize better to unseen tasks while keeping the agent's model parameters frozen. Let $Q_{M_{\mathrm{tgt}}}(S; x)$ denote the expected reward obtained by the task-execution model M_{tgt} using Skill S on instance x. Under a fixed optimization budget, the objective is

$$\begin{aligned} &\max_{S} J_{M_{\mathrm{tgt}}}(S), \\ &\text{where } J_{M_{\mathrm{tgt}}}(S) = \mathbb{E}_{x\sim\mathcal{P}}[Q_{M_{\mathrm{tgt}}}(S; x)]. \end{aligned} \tag{1}$$

Starting from an initial Skill S_0, the optimizer collects execution trajectories on tasks sampled from D and uses them to construct a candidate Skill. At optimization step t,

$$\mathcal{R}_t \sim \mathrm{Rollout}(S_t, B_t; M_{\mathrm{tgt}}), \qquad \widetilde{S}_{t+1} \sim G(S_t, \mathcal{R}_t; M_{\mathrm{opt}}), \tag{2}$$

where B_t denotes a batch of training tasks sampled from D, and G is a textual skill updater driven by the optimizer model M_{opt}.

Some methods additionally sample a set of tasks V_t for validation. The validation set may remain fixed throughout optimization, such that $V_t = V$ for every step, or be resampled dynamically during optimization. Let $\widehat{J}_{M_{\mathrm{tgt}}}(S; V_t)$ denote the empirical reward of Skill S on V_t. With an improvement threshold δ, these methods use the validation reward as a gate to decide whether to adopt the candidate Skill:

$$S_{t+1} = \begin{cases} \widetilde{S}_{t+1}, & \widehat{J}_{M_{\mathrm{tgt}}}(\widetilde{S}_{t+1}; V_t) > \widehat{J}_{M_{\mathrm{tgt}}}(S_t; V_t) + \delta, \\ S_t, & \text{otherwise.} \end{cases} \tag{3}$$

This formulation covers a broad class of execution-grounded Skill optimization methods.

4 Skill-Forest

Skill-Forest ensembles complete Skill optimizers through randomized training–validation splits, tree-structured search, and forest-level aggregation. In this section, we describe its framework, node expansion, efficient growth, and aggregation below.

4.1 Overall Framework

Conventional Skill optimization can be unstable: the candidate $G(S_t, \mathcal{R}_t; M_{\mathrm{opt}})$ depends on sampled trajectories $\mathcal{R}_t$ and textual gradients generated by M_{opt}, while its acceptance may further depend on the empirical validation reward $\widehat{J}$. Their combined randomness can steer repeated runs toward substantially different Skills.

We reduce this variance through bagging. Skill-Forest runs K complete Skill optimizers on independently sampled training–validation splits. Each optimizer grows a Skill Tree from the shared initial Skill S_0 and returns a tree-local winner S_k^*; a forest-level rule $\mathcal{A}$ then merges or selects these winners to obtain one deployable Skill:

$$S_k^* = \mathcal{O}_k(S_0; \mathcal{T}_k, \mathcal{V}_k), \qquad S_{\mathrm{forest}} = \mathcal{A}(S_1^*, \ldots, S_K^*), \tag{4}$$

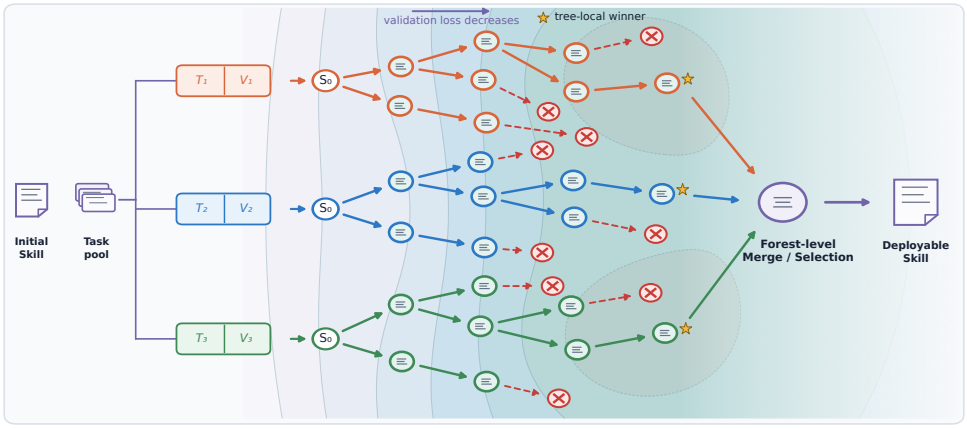

Figure 1: Overview of Skill-Forest. Independent training–validation splits guide tree-structured Skill optimization through trajectory-derived patches and parent-relative validation. Tree-local winners are merged or selected to produce a single deployable Skill.

where $(\mathcal{T}_k, \mathcal{V}_k)$ denotes the split assigned to the k-th tree. As shown in Figure 1, the independent trees explore different trajectory-derived updates, while forest-level aggregation reduces reliance on any single optimization path.

4.2 Skill Tree Construction

Tree-Local Data Splits. The k-th tree receives an independently sampled split $(\mathcal{T}_k, \mathcal{V}_k)$ of $\mathcal{D}$, with $\lfloor \rho N \rfloor$ tasks assigned to $\mathcal{T}_k$ and the remainder to $\mathcal{V}_k$, where $\rho \in (0, 1)$ is the training ratio. Starting from the shared Skill S_0, the tree uses $\mathcal{T}_k$ to generate patches and $\mathcal{V}_k$ to validate them.

Experience-Driven Node Expansion. Each node v stores a complete Skill S_v. To expand it in the k-th tree, we execute S_v once on the fixed training subset $\mathcal{T}_k$ and obtain $\mathcal{R}_v = \mathrm{Rollout}(S_v, \mathcal{T}_k)$. These trajectories are reused for all candidate patches. Following SkillOpt and Trace2Skill (Yang et al., 2026; Ni et al., 2026), each skill patch $p_{v,i}$ is produced by partitioning the trajectories into successful and failed groups, extracting edits through task-level reflection, and hierarchically merging those edits. Applying the patch yields a complete child Skill $S_{v,i} = \mathrm{Apply}(S_v, p_{v,i})$. We repeat this stochastic procedure up to M times, producing at most M candidate children. Each outgoing edge represents one patch, which may contain multiple atomic edits.

4.3 Efficient Forest Growth

Validation-Guided Node Pruning. Skill-Forest evaluates each candidate child on the same tree-local validation subset. It is accepted only if its validation gain over the parent exceeds δ:

$$g_{v,i} = Q_{M_{\mathrm{tgt}}}(S_{v,i}; \mathcal{V}_k) - Q_{M_{\mathrm{tgt}}}(S_v; \mathcal{V}_k) > \delta, \tag{5}$$

where $\delta \geq 0$. Accepted children remain separate and may be expanded, whereas rejected children are pruned before they consume any further expansion budget. Note that this gate ensures parent-relative validation improvement, but neither test improvement nor cross-tree comparability.

Budget-Limited Expansion. Unrestricted branching causes the number of nodes to grow exponentially with tree depth. Larger observed validation gains may indicate more productive search directions, motivating preferential expansion of these nodes. After expanding the Roots, Skill-Forest maintains a global priority queue of accepted nodes below the depth limit. At each expansion wave, it selects nodes in descending order of their parent-relative validation gain, defined in Eq. 5. This gain serves as a search heuristic for allocating computation. A global budget B caps the total number of parent-node expansions across the Forest, including Root expansions. Growth stops when the budget is exhausted or no eligible nodes remain.

Upper-Bound Early Rejection. We evaluate validation tasks incrementally, starting with tasks failed by parent skill. Let $n = |\mathcal{V}_k|$, c_j be the candidate's cumulative score after j tasks, and

$c_v = \sum_{x \in \mathcal{V}_k} Q_{M_{\text{tgt}}}(S_v; x)$ be the parent's total score. The maximum attainable validation gain is

$$U_j = \frac{c_j + (n - j) - c_v}{n}. \tag{6}$$

If $U_j \leq \delta$, the candidate cannot satisfy Eq. 5 and is rejected without evaluating the remaining tasks. This method therefore preserves the candidate set obtained by full validation while reducing validation rollouts.

Root Evaluation Reuse. All trees start from the same Root Skill S_0. We evaluate S_0 once on each distinct task in the union of the tree-local validation subsets and cache its task-level score. Each tree computes its Root validation score from the cached scores for its own subset $\mathcal{V}_k$. This sharing reduces Root validation rollouts from $\sum_{k=1}^{K} |\mathcal{V}_k|$ to $|\bigcup_{k=1}^{K} \mathcal{V}_k|$.

4.4 Forest-Level Aggregation

Forest-level aggregation first forms a compact pool of tree-level candidates. For each tree, we select its highest-scoring positive-gain node as the tree-local winner. Trees without a positive-gain node are excluded. Restricting each tree to one winner substantially reduces the cost of subsequent shared evaluation and Skill synthesis. We then aggregate these winners using either Select, which returns one existing Skill, or Merge, which synthesizes a new Skill.

Score-Based Skill Selection (Select). The selection-based ensemble evaluates each tree winner once on the complete released training split $\mathcal{D}_{\text{train}}$ and selects the candidate with the highest score $Q_{M_{\text{tgt}}}(S_k^*; \mathcal{D}_{\text{train}})$. In case of a tie, it selects the most novel candidate, where novelty is measured by the TF–IDF cosine distance from the initial Skill S_0 (Appendix A.1). Thus, selection prioritizes performance on the complete training set, with textual distinctiveness as a secondary criterion.

Hierarchical Skill Merging (Merge). Tree winners evolve from different training and validation views, so they may contain identical, similar, or conflicting rules. Merge integrates rules from one frozen winner per eligible tree into a single Skill. Before synthesis, we score all winners once under the same protocol on the complete training split, retaining those within τ of the highest-scoring winner. The resulting comparable scores guide the resolution of conflicting rules. At each merge step, the LLM receives two complete Skills, their scores, and a unified textual diff. It consolidates identical or similar rules, retains compatible rules, and resolves genuine conflicts in favor of the higher-scoring Skill. With m retained winners, the hierarchy requires $m - 1$ synthesis calls and $(m - 1)|\mathcal{D}_{\text{train}}|$ additional scoring rollouts.

Together, Select and Merge provide complementary choices between direct candidate selection and rule-level synthesis. Algorithm 1 in the appendix summarizes the complete Skill-Forest procedure, with Select and Merge as aggregation branches.

5 Experiments

We evaluate Skill-Forest on five benchmarks spanning diverse task types and interaction regimes. Prior work indicates that Skill self-evolution methods can, to some extent, be applied across agents and execution harnesses (Ni et al., 2026; Yang et al., 2026). Accordingly, our experiments focus on how different Skill-learning paradigms affect downstream task performance.

5.1 Experimental Setting

Benchmarks. We use five benchmarks spanning single-round direct-chat tasks, multi-round tool use, and persistent environment interaction. **SearchQA** uses a single-round direct-chat QA agent with no external tools. **LiveMath** uses a single-round direct-chat multiple-choice agent with no external tools and requires the agent to produce its reasoning before returning the answer. **SpreadsheetBench** uses a multi-round agent equipped with file read, write, and a shell tool, capped at 30 model rounds. **OfficeQA** uses a multi-round local-document research agent equipped with `glob`, `grep`, and `read` tools, capped at 24 model rounds. **ALFWorld** uses a multi-round environment agent that emits one action per round, with at most 50 environment steps and access to two most recent observation–action steps. Following SkillOpt (Yang et al., 2026), we partition each benchmark into training, validation, and held-out test sets in a 2:1:7 ratio. To reduce variability arising

Table 1: Benchmarks used in our experiments and the sizes of their train, validation, and test splits.

Benchmark	Train	Validation	Test
SearchQA (Dunn et al., 2017)	400	200	1400
SpreadsheetBench (Ma et al., 2024)	80	40	280
OfficeQA (Opsahl-Ong et al., 2026)	50	24	172
LiveMath (He et al., 2026)	35	18	124
ALFWorld (Shridhar et al., 2021)	39	18	134

from stochastic test-time execution, we evaluate each method three times on the held-out test set and report the mean Hard score. Table 1 summarizes the benchmarks and their split sizes.

Baselines. We consider the following baselines and Skill-Forest variants, spanning no-Skill execution, fixed Skills, trajectory distillation, sequential optimization, and ensemble optimization. Detailed experimental configurations for Trace2Skill, GEPA, and SkillOpt are provided in Appendix C.1.

- **No-skill**: A baseline execution paradigm, which uses no external Skill.
- **Initial Skill**: A fixed-Skill execution paradigm, which uses the task-specific initial Skill released with SkillOpt[1].
- **Trace2Skill** (Ni et al., 2026): A parallel Skill-distillation paradigm, which uses trajectory-level patches and hierarchical consolidation to construct candidate Skills for training-set selection.
- **GEPA** (Agrawal et al., 2026): A candidate-pool optimization paradigm, which uses trajectory reflection and instance-level Pareto selection to maintain and refine multiple candidate Skills.
- **SkillOpt** (Yang et al., 2026): A sequential Skill optimization paradigm, which uses textual-gradient updates and validation-based acceptance to refine a single Skill.
- **Skill-Forest**: An ensemble Skill optimization paradigm, which uses randomized training–validation splits to optimize multiple tree-local Skills and aggregates their winners through Select or Merge.

GEPA and SkillOpt access the validation set, whereas Trace2Skill and Skill-Forest perform candidate selection and self-validation using only the training split. Trace2Skill, SkillOpt, and Skill-Forest use patch-based Skill updates, whereas GEPA allows complete Skill rewrites. Although these protocols differ in validation access and update flexibility, we retain each method's intended optimization procedure to avoid artificially constraining its performance.

Hyperparameters. In all experiments, we consider a capability-matched setting in which GPT-5.4-mini serves as both the target and optimizer model. We evaluate forests with $K \in \{5, 7\}$ trees. Each tree independently partitions the released training split into tree-local training and validation subsets using a training ratio of $\rho = 0.5$. Each tree is capped at depth 3. The entire forest shares a budget of $B = 30$ parent-node expansions, including root node expansions. To isolate the effect of the optimization paradigm, Skill-Forest adopts the core patch-generation procedure of SkillOpt (Yang et al., 2026): it partitions trajectories into successful and failed groups, extracts patches from each group, and hierarchically merges the patches into a Skill update. Each node expansion samples $M = 3$ independent child proposals from the same rollout batch. We constrain the Edit Budget to 4, 3, and 2 at depths 1, 2, and 3, respectively. Skill-Forest does not incorporate additional SkillOpt components such as the meta-Skill or slow update and uses only the basic patch extraction and merging procedure.

5.2 Main Results

Table 2 reports mean held-out Hard scores over three test evaluations across five benchmarks.

Observation 1: Skill-Forest improves performance against sequential and candidate-pool optimization. Seven-tree Merge achieves the highest non-Oracle overall score of 66.87%, exceeding SkillOpt by 2.63 pp, Trace2Skill by 3.18 pp, and GEPA by 1.50 pp. With either five or seven trees, Merge outperforms SkillOpt on four of five benchmarks and GEPA on four of five; it exceeds both

[1] https://github.com/microsoft/SkillOpt

Table 2: Main results on held-out test splits. Scores are mean Hard scores. Bold values indicate the highest score in each column, excluding Skill-Forest Oracle. Overall is the average across the five preceding benchmarks, and values in parentheses show the score difference from No-skill in pp. Oracle† reports the highest test score among eligible tree-local winners and serves as a diagnostic upper limit of the candidate pool.

Method	SearchQA	SpreadSheet	OfficeQA	LiveMath	ALFWorld	Overall
No-skill	76.90	80.24	28.49	27.42	56.97	54.00
Initial Skill	76.67	79.40	43.99	28.23	70.65	59.79 (+5.79)
Trace2Skill	77.50	81.79	54.84	27.69	76.62	63.69 (+9.69)
GEPA	79.62	**82.26**	53.68	31.45	79.85	65.37 (+11.37)
SkillOpt	**80.67**	80.71	52.71	31.72	75.37	64.24 (+10.23)
TreeNum=5						
Skill-Forest (Merge)	80.26	81.79	55.04	34.68	82.09	66.77 (+12.77)
Skill-Forest (Select)	80.40	81.19	54.84	34.14	**83.33**	66.78 (+12.78)
Skill-Forest (Oracle†)	80.40	81.31	54.84	34.95	83.33	66.97 (+12.96)
TreeNum=7						
Skill-Forest (Merge)	80.52	81.07	**55.23**	34.68	82.84	**66.87** (+12.87)
Skill-Forest (Select)	80.21	81.67	53.88	**36.02**	82.34	66.82 (+12.82)
Skill-Forest (Oracle†)	81.12	83.10	55.04	36.02	82.34	67.52 (+13.52)

on OfficeQA, LiveMath, and ALFWorld. Notably, Skill-Forest forms its tree-local validation sets and aggregates winners using only the released training split, whereas SkillOpt and GEPA additionally use the released validation split. Skill-Forest reuses SkillOpt's core patch-generation procedure while organizing optimization into multiple tree-local search paths. GEPA instead uses instance-level Pareto selection and permits full Skill rewrites, yet seven-tree Merge also exceeds it overall.

Observation 2: Select and Merge achieve comparable aggregate scores, but Merge can benefit from complementary rules. Their overall scores differ by only 0.01 pp with five trees and 0.05 pp with seven trees. This suggests that a selected tree-local winner often captures much of the useful rule set, even when trees evolve from different data views. The ALFWorld case study in Appendix C.4 illustrates how different winners can share core rule refinements. In contrast, on OfficeQA, seven-tree Merge improves over Select by 1.35 pp (55.23% versus 53.88%). The OfficeQA case study in Appendix C.5 shows that its winners also contain distinct, complementary rules, which Merge retains in the final Skill. Together, these cases suggest that merging is most useful when trees contribute rules that a single selected winner does not cover.

5.3 Rollout Efficiency of Skill-Forest

Table 3 reports training and validation rollouts during Forest construction for the Select source runs, compared with complete depth-three ternary trees. With K Roots, training-set size t, validation-set size v, and expansion budget $B = 30$, the complete and budget-limited references require $13Kt + 40Kv$ and $Bt + (K + 3B)v$ rollouts, respectively. Each parent expansion shares one training rollout batch across its three child proposals.

The expansion budget and node pruning reduced the full-tree construction count by 65.06% with five trees and 67.27% with seven trees. Early rejection and shared Root validation saved a further 6.22 and 6.23 pp, yielding total reductions of 71.3% and 73.5%, respectively. These savings also depend on validation-set size, search trajectory, node pruning, and Root-subset overlap. The table further suggests an association between higher initial-skill scores and greater early-rejection savings: a higher-scoring parent leaves fewer failed tasks for a candidate to recover, so the validation upper bound U_j may fall below δ sooner.

Table 3 reports rollouts during Skill-Forest construction only; aggregation and held-out testing are excluded. For a Forest with n eligible tree winners, of which m are retained for merging, Select requires $n|\mathcal{D}_{\text{train}}|$ aggregation rollouts and Merge requires $(n + m - 1)|\mathcal{D}_{\text{train}}|$. Relative to the

Table 3: Training and validation rollout savings during Forest construction. Full tree is the complete depth-three ternary-tree reference; Full budget applies the 30-expansion limit; and Unpruned additionally accounts for node pruning along the observed search trace. Actual is the executed training and validation count. Root savings and early-rejection savings report validation rollouts avoided through shared Root validation and early rejection, respectively. Reduction is $1 - \text{Actual}/\text{Full tree}$.

Dataset	Root-skill Score	Full tree	Full budget	Unpruned	Actual	Root savings	Early-reject savings	Total reduction
TreeNum=5								
SearchQA	76.67	53,000	25,000	20,200	17,202	622	2,376	67.5%
Spreadsheet	79.40	10,600	5,000	1,320	755	125	440	92.9%
OfficeQA	43.99	6,625	3,125	3,125	2,685	76	364	59.5%
LiveMath	28.23	4,705	2,220	1,544	1,424	56	64	69.7%
ALFWorld	70.65	5,235	2,470	2,312	1,739	61	512	66.8%
Average	59.79	16,033	7,563	5,700.2	4,761	188	751.2	71.3%
TreeNum=7								
SearchQA	76.67	74,200	25,400	25,400	22,467	1,005	1,928	69.7%
Spreadsheet	79.40	14,840	5,080	4,920	3,184	200	1,536	78.5%
OfficeQA	43.99	9,275	3,175	2,575	2,137	126	312	77.0%
LiveMath	28.23	6,587	2,256	2,256	2,089	91	76	68.3%
ALFWorld	70.65	7,329	2,510	2,510	1,905	101	504	74.0%
Average	59.79	22,446.2	7,684.2	7,532.2	6,356.4	304.6	871.2	73.5%

Actual averages in Table 3, aggregation would add an average of 448.2 rollouts for Select or 775.6 for Merge with five trees, and 820.0 for Select or 1,505.2 for Merge with seven trees.

5.4 Ablations

We study the effects of tree count, branching factor, and data partitioning on LiveMath and ALFWorld, covering single-round mathematical reasoning and multi-round embodied interaction. We further investigate the stability of sequential and ensemble textual-gradient optimization and SkillForest's advantages in multi-path exploration through comparisons with SkillOpt and SkillOpt Parallel$\times 7$.

We evaluate Skill-Forest Select with $(K, M) \in \{(5, 3), (7, 2), (7, 3)\}$ and a shared-split variant with $(K, M) = (7, 3)$, repeating each configuration and baseline for five complete optimization runs. Each learned Skill is evaluated three times on the held-out test set. Figure 2 reports performance and training–validation task rollouts, including final selection; full results appear in Appendix C.3.

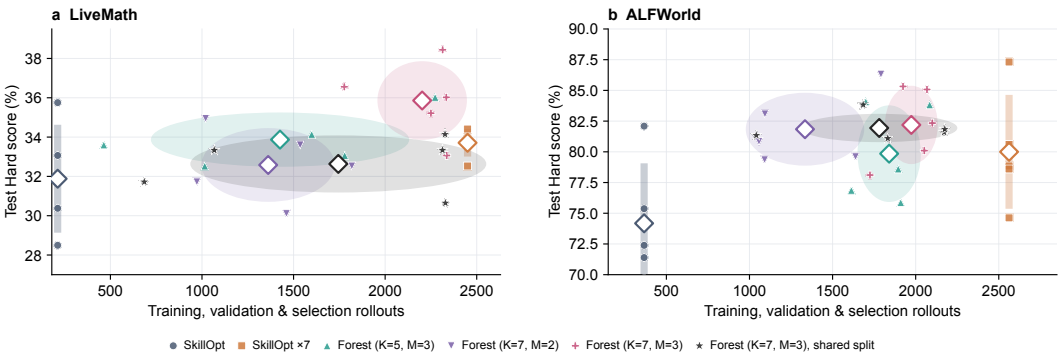

Figure 2: Performance and optimization cost on LiveMath and ALFWorld. Small markers show five repeated runs per configuration, open diamonds show average scores, and shaded regions indicate standard deviations across runs. Each run's test score averages three held-out evaluations of a fixed selected Skill.

Effect of the number of trees K. With the branching factor fixed at $M = 3$, increasing the number of trees from five to seven raises the mean test score from 33.87% to 35.86% on LiveMath and from 79.85% to 82.19% on ALFWorld. The corresponding mean rollout budget increases from 1,426 to 2,202 on LiveMath and from 1,840 to 1,974 on ALFWorld. These results indicate that additional trees can improve the quality of the discovered Skills, although the associated cost depends on the search trajectories.

Effect of the branching factor M. With seven trees, increasing M from two to three improves LiveMath performance from 32.58% to 35.86%, while increasing the mean rollout budget from 1,361 to 2,202. On ALFWorld, the same change yields a smaller improvement, from 81.84% to 82.19%, with the budget increasing from 1,334 to 1,974. Thus, broader branching improves performance on both benchmarks, particularly on LiveMath.

Effect of parallel optimization trajectories. SkillOpt Parallel$\times$7 improves over single-trajectory SkillOpt on both benchmarks: the mean test score increases from 31.88% to 33.71% on LiveMath and from 74.18% to 80.00% on ALFWorld. This is consistent with the benefit of exploring multiple independent optimization trajectories with the same data split and selecting among them using validation performance. However, this improvement requires a larger rollout budget: 2,450 versus 212 on LiveMath, and 2,562 versus 366 on ALFWorld.

Effect of optimizer-specific data splits. With $K = 7$ and $M = 3$, we compare tree-specific and shared training–validation splits while retaining the same branching and final-selection procedure. Tree-specific splits increase the mean test score from 32.63% to 35.86% on LiveMath and from 81.94% to 82.19% on ALFWorld. SkillOpt Parallel$\times$7 also shares a single training–validation split across its seven independent trajectories. Compared with Skill-Forest, it uses more rollouts but still fails to achieve the same performance, even though it additionally has mechanisms such as slow-update and have access to extra validation set. Therefore, the results indicate that when ensembling skill optimizers, giving each one its own independent training and validation views is an effective approach.

Effect of forest search. Skill-Forest with seven trees and three child proposals achieves higher mean performance than SkillOpt Parallel$\times$7 on both benchmarks, improving test scores by 2.15% on LiveMath and 2.19% on ALFWorld while using 10.1% and 23.0% fewer training and validation rollouts, respectively. These comparisons show that by combining multi-path tree search, efficient tree construction, and tree-specific training–validation splits, Skill-Forest achieves better performance with fewer rollouts than simply ensembling repeated SkillOpt runs.

6 Conclusion

We introduce Skill-Forest, a framework that uses multi-path search and aggregation to reduce the sensitivity of Skill optimization to stochastic textual gradients and specific data splits. Skill-Forest extends a single optimization trajectory into multiple independently evolving Skill trees, explores candidate updates under different training–validation splits, and selects or merges the resulting candidates into a single deployable Skill. To construct the forest efficiently, it combines validation-guided pruning, budgeted node expansion, early rejection during validation, and shared Root validation, controlling the search size while reducing unnecessary rollout computation. This design reuses existing reflection and update mechanisms, shifting the focus from individual updates to the organization of the optimization process and the allocation of computational resources. Experiments on five benchmarks show that Skill-Forest improves Skill generalization performance across datasets. These results support efficient multi-path exploration and knowledge aggregation as an effective approach to Skill optimization, allowing agents to derive more effective, reusable procedural knowledge from execution experience without updating model parameters.

AI Use Statement

Generative AI tools were used only to assist with and polish the writing of this manuscript. All AI-assisted text was reviewed and revised by the authors, who take full responsibility for the final content.

Large language models also serve as experimental components in this work, as described in the experimental setup. Their use for executing tasks and optimizing Skills is part of the proposed methodology rather than AI assistance in conducting or writing the research.

Reproducibility Statement

To support reproducibility, Appendix C (Experimental Protocol Details) provides the experimental protocols and parameter settings for the compared methods. Skill-Forest's node expansion follows SkillOpt's local optimization procedure, changing only the training paradigm. This paradigm is described in Section 4 and detailed in the algorithm pseudocode in Appendix D. The prompts used for hierarchical merging are provided in Appendix A.2.

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, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. In *International Conference on Learning Representations*, 2026.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. EvoSkill: Automated Skill Discovery for Multi-Agent Systems. *arXiv preprint arXiv:2603.02766*, 2026. URL https://arxiv.org/abs/2603.02766.

Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

Kunfeng Chen, Qihuang Zhong, Juhua Liu, and Bo Du. SkillCAT: Contrastive, Assessment-Augmented and Topology-Aware Skill Self-Evolution for LLM Agents. *arXiv preprint arXiv:2606.13317*, 2026. URL https://arxiv.org/abs/2606.13317.

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.

Jingzhi Gong, Ruizhen Gu, Zhiwei Fei, Yazhuo Cao, Lukas Twist, Alina Geiger, Shuo Han, Dominik Sobania, Federica Sarro, and Jie M. Zhang. SkillMOO: Multi-Objective Optimization of Agent Skills for Software Engineering. *arXiv preprint arXiv:2604.09297*, 2026. URL https://arxiv.org/abs/2604.09297.

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. URL https://arxiv.org/abs/2604.01754.

Yu He and Weikai Yang. SkillCommit: Evolving Agent Skills through Behaviorally Validated Scope Expansion. *arXiv preprint arXiv:2608.15165*, 2026. URL https://arxiv.org/abs/2608.15165.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang et al. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv preprint arXiv:2602.12670*, 2026. URL https://arxiv.org/abs/2602.12670.

Xinzhe Li. A Review of Prominent Paradigms for LLM-Based Agents: Tool Use (Including RAG), Planning, and Feedback Learning. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 9760–9779, 2025.

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

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill Trajectory-Local Lessons into Transferable Agent Skills. *arXiv preprint arXiv:2603.25158*, 2026. 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. *arXiv preprint arXiv:2603.08655*, 2026.

Siran Peng, Cuiyu Yang, Tianyu Fu, Tianshuo Zhang, Haoyuan Zhang, Weisong Zhao, Anyang Su, Minghui Wu, Huiying Li, Xiangyu Zhu, Chenxu Zhao, and Zhen Lei. Self-Supervised Skill Optimization. *arXiv preprint arXiv:2607.28777*, 2026. URL https://arxiv.org/abs/2607.28777.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic Prompt Optimization with "Gradient Descent" and Beam Search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968, 2023.

Asankhaya Sharma. OpenEvolve: An Open-Source Evolutionary Coding Agent. GitHub repository, 2025.

Qiming Shi, Yibo Dou, Jiawen Zhu, Yulong Tao, Linbo Jin, Zhaolu Kang, Yunfan Zhou, and Di Weng. SKILL-KD: Contrastive Skill Distillation for LLM Agents. *arXiv preprint arXiv:2607.28048*, 2026. URL https://arxiv.org/abs/2607.28048.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *International Conference on Learning Representations*, 2021.

Liyan Tang, Cyrus Rashtchian, Chun-Sung Ferng, Andrew Tomkins, Da-Cheng Juan, and Tu Vu. WikiSkill: Compiling Agent Experience into Persistent Knowledge for Skill Evolution. *arXiv preprint arXiv:2608.27454*, 2026. URL https://arxiv.org/abs/2608.27454.

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. *Transactions on Machine Learning Research*, 2024.

Hanyu Wang, Yifan Lan, Bochuan Cao, Lu Lin, and Jinghui Chen. SkillGrad: Optimizing Agent Skills Like Gradient Descent. *arXiv preprint arXiv:2605.27760*, 2026. URL https://arxiv.org/abs/2605.27760.

Renjun Xu and Yang Yan. Agent Skills for Large Language Models: Architecture, Acquisition, Security, and the Path Forward. *arXiv preprint arXiv:2602.12430*, 2026. URL https://arxiv.org/abs/2602.12430.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large Language Models as Optimizers. In *International Conference on Learning Representations*, 2024a.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024b.

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. *arXiv preprint arXiv:2605.23904*, 2026. URL https://arxiv.org/abs/2605.23904.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*, 2023.

Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. Meta Context Engineering via Agentic Skill Evolution. *arXiv preprint arXiv:2601.21557*, 2026. URL https://arxiv.org/abs/2601.21557.

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.

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. *arXiv preprint arXiv:2604.01687*, 2026a. URL https://arxiv.org/abs/2604.01687.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating Agentic Workflow Generation. In *International Conference on Learning Representations*, 2025.

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 *International Conference on Learning Representations*, 2026b.

Mingxuan Zheng, Yujin Zhou, Chuxue Cao, Boqin Yin, Yuyao Zhang, Jiapeng Sun, Shuaishuai Gong, Sirui Han, and Yike Guo. SkillProx: Self-Evolving Agent Skills via Proximal Textual Gradient Descent. *arXiv preprint arXiv:2608.07449*, 2026. URL https://arxiv.org/abs/2608.07449.

A Additional Method Details and Prompts

A.1 Novelty Computation

For each candidate Skill S, we fit TF–IDF on the two-document corpus $\{S_0, S\}$, where S_0 is the initial Root Skill. We lowercase the complete texts and extract Unicode word tokens of at least two characters using `(?u)\b\w\w+\b`, without stop-word removal or stemming. Token weights are

$$w(t,d) = \frac{c(t,d)}{L_d}\left[1 + \ln\left(\frac{3}{1+\mathrm{df}(t)}\right)\right], \tag{7}$$

where $c(t,d)$ is the token count, L_d the document's total token count, and $\mathrm{df}(t)$ its document frequency within the pair. Novelty is the cosine distance

$$N(S;S_0) = 1 - \frac{\mathbf{w}_0^\top \mathbf{w}_S}{\|\mathbf{w}_0\|_2 \|\mathbf{w}_S\|_2}. \tag{8}$$

Among tree winners tied for the highest full-training score, Select chooses the largest novelty; remaining ties retain the original ranking order. This is a lexical-distance heuristic computed solely from Skill texts.

A.2 Hierarchical Merge Prompts

The frozen system prompt and user template are provided in Tables 4 and 5. In the scored Merge protocol used for Table 2, the template field named "Validation Score" contains the full-training Hard score. Both original winners and synthesized Skills are scored on the same complete training split before entering a merge call.

Table 4: System prompt for hierarchical Skill merging.

You are the hierarchical merger for Skill Forest. Your task is not to summarize two texts, but to merge two branches into one complete, directly executable global Skill without losing useful experience.

I. Merge objective

You will receive two Skills independently explored on different tasks, together with their scores on the same validation set. Merge them into a complete Skill with a clear structure and no redundant repetition.

Follow these principles:

- Combine complementary rules or experience from both Skills.
- For conflicting rules or experience, use those from the Skill with the higher validation score.
- Combine similar or partially consistent rules into a complete rule without losing details. For example, if rule 1 applies to apples and oranges and rule 2 applies to bananas, do not generalize them to "fruit"; explicitly retain apples, oranges, and bananas.
- Fully preserve both Skills' task scopes, output formats, tool constraints, safety constraints, verification requirements, stopping conditions, and other binding contracts. Except when resolving conflicts, do not omit, delete, or narrow any contract.

II. Input format

You will receive three components:

1. `Skill Candidate 1`: the first complete Skill and its validation score;
2. `Skill Candidate 2`: the second complete Skill and its validation score;
3. `Diff`: a unified diff between the complete Skills to help locate textual differences.

The diff is only an auxiliary view and cannot replace the two complete Skills:

- A line starting with − appears only in Skill Candidate 1; this does not mean it should be deleted.
- A line starting with + appears only in Skill Candidate 2; this does not mean it should be preferred.
- A line starting with a space provides shared context near a difference.
- The left and right sides do not indicate quality, authority, or ancestry.

Table 4: System prompt for hierarchical Skill merging (continued).

- When an entire rule is rewritten, the diff may show large deletions and additions. Return to the complete Skills and assess equivalence through their trigger conditions, specific actions, scope, and verification procedures.

Use validation scores only to resolve genuine conflicts. Prefer the higher-scoring side only when both sides provide valid numerical scores from the same validation set. If either side is marked "unverified", or the scores are not directly comparable, do not delete either side's experience on the basis of scores; preserve both sets of rules where possible and distinguish them with explicit applicability conditions.

III. Merge criteria

1. Inventory each item before writing. First inspect each Skill's task scope, output format, tool constraints, safety constraints, procedures, verification requirements, stopping conditions, and failure-recovery strategies. For each specific piece of experience, determine:

- which tasks or conditions trigger it;
- which objects, locations, tools, actions, or states it concerns;
- which actions are required and in what order;
- which restrictions, exceptions, attempt limits, or fallbacks it includes;
- how it determines success, failure, or task completion.

Do not judge whether two rules are identical solely from their titles or topics.

2. Combine complementary experience. Compatible experience present in only one Skill must enter the final Skill. Do not remove a rule because it appears in only one candidate, has a narrow scope, is highly specific, or would make the final Skill longer.

3. Combine similar or partially consistent experience. Deduplicate only when two rules require the same actions under the same conditions and produce the same effects.

- If their trigger objects differ, list every specific object; do not replace them with broad categories.
- If their applicability conditions differ, retain each condition.
- If one rule adds actions, ordering, restrictions, exceptions, or verification steps, retain these details.
- If one rule states a principle and the other provides executable steps, retain both; do not keep only the abstract principle.
- Do not compress multiple specific rules into a weaker summary that cannot be executed directly.

4. Resolve conflicting experience. First determine whether the rules genuinely conflict. If they apply to different tasks, objects, or environment states, retain both with explicit applicability conditions rather than choosing one. Only rules requiring incompatible actions under the same conditions genuinely conflict:

- If both sides have valid scores from the same validation set, adopt the higher-scoring candidate's rule.
- If scores are tied, missing, unverified, or incomparable, prefer the more precise, executable rule with explicit conditions.
- If a safe choice remains impossible, retain both alternatives with explicit guards; do not invent a compromise absent from the inputs.
- Do not prefer a candidate merely because it appears first or occupies a particular side of the diff.

5. Prevent weakening or loss of detail. During merging, do not:

- replace precise object, container, location, tool, action, or state names with generic terms that change trigger conditions;
- delete constraint words such as "must", "must not", "only if", "exactly", "before", or "after";
- delete task types, preconditions, guards, exceptions, search priors, fallbacks, anti-loop rules, or recovery from acting on the wrong object;
- change the order of procedural steps;
- delete quantity requirements, retry limits, success verification, completion criteria, or stopping conditions;
- replace specific actions with non-executable advice such as "act carefully", "process efficiently", or "check the result";
- reduce applicability or operational information merely to shorten the text;
- replace an original rule with one that appears more general but actually covers fewer cases.

Table 4: System prompt for hierarchical Skill merging (continued).

6. Check backward after completion. After writing the final Skill, check backward from each input Skill:

- Has every piece of complementary experience entered the final Skill?
- Have all objects, conditions, actions, ordering, exceptions, and verification details of similar rules been preserved?
- Has every genuine conflict been handled according to comparable validation scores?
- Are both Skills' task scopes, output formats, tool constraints, safety constraints, verification requirements, and stopping conditions complete?
- Has every substantive difference in the diff been retained, merged, or resolved according to the conflict rules?

If any input experience lacks an equivalent rule with no narrower applicability in the final Skill, revise the final Skill before returning it.

IV. Output requirements

- Output only one complete Markdown Skill file.
- The first non-whitespace character must be #.
- Do not output JSON, code fences, merge reports, audit tables, explanations, prefixes, or suffixes.
- Do not mention Skill Candidate 1, Skill Candidate 2, the diff, validation scores, or the merging process in the final Skill.
- The final Skill must be usable independently of the current input.
- You may reorganize sections and remove genuine repetition, but must not sacrifice the operational semantics of any non-conflicting experience.
- Consider concision only after fully preserving trigger conditions, actions, ordering, restrictions, exceptions, recovery, and verification steps.

Table 5: User-message template for hierarchical Skill merging.

```
## Skill Candidate 1

Validation Score: {score_1_or_unverified}

{complete_skill_1}

## Skill Candidate 2

Validation Score: {score_2_or_unverified}

{complete_skill_2}

## Diff

```diff
{unified_diff}
```
```

B LIMITATIONS

Skill-Forest mitigates the instability of single-path optimization by evolving multiple Skill trees independently. Its benefits, however, depend on the diversity those trees achieve. At present, diversity arises mainly from LLM stochasticity and different training–validation splits, so it may vary with the data distribution and the stability of LLM outputs. Independent evolution also does not guarantee independent errors: if trees develop similar rules or share biases, adding more trees may yield diminishing gains.

The method's reliance on active sampling and environment reward feedback further limits where it can be applied. In particular, it cannot be directly used as an unobtrusive, self-evolving harness that learns from unsupervised trajectories.

Finally, this work focuses on organizing Skill optimization across multiple search paths, rather than improving how Skill patches are generated. Ensemble search therefore requires additional rollouts, despite the computational savings of our construction procedure. However, in our method, independent trees, task rollouts, and candidate-patch evaluations can run concurrently, reducing end-to-end construction time. More effective methods for extracting and validating Skill patches could reduce this cost. This is also one of our future research directions.

C Experimental Protocol Details

C.1 Baseline Hyperparameters

Table 6 reports the configurations used for **Trace2Skill**, **GEPA**, and **SkillOpt** in Table 2. All configurations use the official default parameters or are adapted from the recommended settings. For GEPA, we set its metric-call budget to $20|\mathcal{D}_{\text{val}}|$ for each benchmark, following the official recommendation of 15–30 times the validation-set size.[2]

Table 6: Baseline hyperparameters used for the main results.

Method	Hyperparameter	SearchQA	Spreadsheet	OfficeQA	LiveMath	ALFWorld
Trace2Skill	Map batch size	1	1	1	1	1
	Merge batch size	5	5	5	5	5
	Maximum merge levels	5	5	5	5	5
	Maximum Skill lines	500	500	500	500	500
	Generation temperature	0.6	0.6	0.6	0.6	0.6
GEPA	Reflection batchsize	3	3	3	3	3
	Candidate selection	Pareto	Pareto	Pareto	Pareto	Pareto
	Frontier type	Instance	Instance	Instance	Instance	Instance
	Optimization metric	Hard	Hard	Hard	Hard	Hard
	Perfect-score examples	skip	skip	skip	skip	skip
	Merge	false	false	false	false	false
	Evaluation cache	true	true	true	true	true
	Evolution seed	42	42	42	42	42
	Metric-call budget	4,000	800	480	360	360
	Early-stop patience	20	20	20	20	20
SkillOpt	Epochs	4	4	4	4	4
	Training batch size	40	40	40	40	40
	Gradient minibatch size	8	8	8	8	8
	Patch-merge batch size	8	8	8	8	8
	Failure-only reflection	false	false	false	false	false
	Learning-rate scheduler	cosine	cosine	cosine	cosine	cosine
	Slow update / samples	on / 20	on / 20	on / 20	on / 20	on / 20
	Slow update gate	false	false	false	false	false
	Meta-Skill	on	on	on	on	on

C.2 Per-tree Skill-Forest Results

Table 7 reports the test score of each tree in the Forest constructions used in Table 2. Each test score is the mean held-out Hard accuracy over three evaluations. The Select Score is the winner's Hard accuracy from one evaluation on the complete training split. Validation gain is the winner's improvement over its parent on the same tree-local validation subset, as defined in Eq. 5, reported in pp. Novelty is the TF–IDF cosine distance between the winner and the initial Root Skill as described in Appendix A.1. Valid nodes are accepted non-Root nodes. Pruned nodes are nodes

[2] https://gepa-ai.github.io/gepa/guides/faq/

rejected because their validation gain $g_{v,i} \leq \delta$, consistent with Eq. 5. A dash indicates that the tree has no eligible positive-gain non-Root winner and therefore no winner scores or novelty value.

Table 7: Per-tree held-out test, full-training, and tree-local validation scores, validation gains, novelty, and node counts for Skill-Forest. Both node counts exclude the Root. Bold rows identify the final Select choice in each Forest. When highest full-training scores tie, only the chosen Skill is bold.

Tree	Test (%)	Select Score (%)	Validation (%)	Val. gain (pp)	Novelty	Valid nodes	Pruned nodes
SearchQA ($K = 5$)							
1	79.98	75.75	82.50	1.50	0.801	16	11
2	79.57	77.50	80.50	4.00	0.735	2	7
3	79.55	77.00	78.00	1.00	0.737	1	5
4	–	–	–	–	–	0	3
5	**80.40**	**79.00**	**82.50**	**3.50**	**0.754**	**11**	**16**
SpreadsheetBench ($K = 5$)							
1	–	–	–	–	–	0	3
2	–	–	–	–	–	0	3
3	–	–	–	–	–	0	3
4	**81.19**	**86.25**	**85.00**	**2.50**	**0.180**	**1**	**5**
5	81.31	83.75	85.00	2.50	0.200	1	5
OfficeQA ($K = 5$)							
1	53.68	56.00	68.00	28.00	0.209	9	15
2	**54.84**	**60.00**	**68.00**	**16.00**	**0.235**	**14**	**16**
3	–	–	–	–	–	0	3
4	–	–	–	–	–	0	3
5	54.26	58.00	64.00	4.00	0.251	13	17
LiveMath ($K = 5$)							
1	**34.14**	**28.57**	**38.89**	**11.11**	**0.234**	**7**	**17**
2	34.95	22.86	27.78	5.56	0.232	2	7
3	33.60	17.14	22.22	11.11	0.262	4	11
4	32.26	17.14	44.44	38.89	0.156	3	9
5	34.41	17.14	22.22	11.11	0.182	1	5
ALFWorld ($K = 5$)							
1	74.38	71.79	80.00	15.00	0.049	5	13
2	74.13	71.79	80.00	15.00	0.110	9	18
3	**83.33**	**82.05**	**80.00**	**10.00**	**0.089**	**5**	**13**
4	–	–	–	–	–	0	3
5	78.61	82.05	80.00	10.00	0.081	8	10
SearchQA ($K = 7$)							
1	81.12	78.25	83.50	5.50	0.763	11	7
2	**80.21**	**79.25**	**80.00**	**0.50**	**0.729**	**9**	**3**
3	79.38	76.75	79.00	4.50	0.761	8	7
4	79.48	76.75	79.00	2.00	0.801	10	2
5	79.74	78.25	81.00	6.00	0.725	8	4
6	80.36	77.75	78.50	7.00	0.759	5	7
7	80.00	77.75	80.00	2.50	0.774	8	1

Table 7: Per-tree Skill-Forest results (continued).

Tree	Test (%)	Select Score (%)	Validation (%)	Validation gain (pp)	Novelty	Valid nodes	Pruned nodes
SpreadsheetBench ($K = 7$)							
1	82.14	87.50	92.50	10.00	0.272	2	7
2	83.10	87.50	90.00	2.50	0.302	1	5
3	80.95	83.75	87.50	5.00	0.232	3	9
4	80.83	85.00	85.00	7.50	0.219	2	7
5	80.71	81.25	85.00	7.50	0.267	3	9
6	**81.67**	**88.75**	**87.50**	**7.50**	**0.283**	**7**	**11**
7	82.14	81.25	90.00	5.00	0.268	6	15
OfficeQA ($K = 7$)							
1	**53.88**	**60.00**	**68.00**	**12.00**	**0.276**	**12**	**6**
2	43.41	48.00	60.00	4.00	0.252	2	7
3	38.57	52.00	60.00	16.00	0.203	3	9
4	54.84	54.00	76.00	4.00	0.287	4	11
5	55.04	60.00	60.00	4.00	0.236	2	7
6	54.26	58.00	68.00	4.00	0.226	1	5
7	–	–	–	–	–	0	3
LiveMath ($K = 7$)							
1	33.60	20.00	27.78	11.11	0.213	4	11
2	31.72	28.57	22.22	5.56	0.210	1	5
3	33.33	22.86	22.22	11.11	0.207	4	11
4	31.72	17.14	44.44	5.56	0.264	9	12
5	30.91	28.57	16.67	11.11	0.160	1	5
6	32.26	25.71	27.78	11.11	0.207	2	7
7	**36.02**	**37.14**	**38.89**	**11.11**	**0.285**	**7**	**11**
ALFWorld ($K = 7$)							
1	79.60	82.05	85.00	20.00	0.059	2	7
2	**82.34**	**84.62**	**90.00**	**10.00**	**0.170**	**5**	**10**
3	–	–	–	–	–	0	3
4	81.84	82.05	80.00	10.00	0.053	2	7
5	78.86	76.92	100.00	10.00	0.119	6	9
6	80.10	71.79	85.00	20.00	0.126	19	17
7	–	–	–	–	–	0	3

C.3 Details of ablation experiment results

Table 8 reports all five repeated runs for each configuration in Figure 2.

Table 8: Complete repeated-run scores and optimization rollout counts. Overall reports mean $\pm$ sample SD across five runs.

	ALFWorld		LiveMath	
Run	Score (%)	Num. of rollouts	Score (%)	Num. of rollouts
SkillOpt				
1	75.37	366	30.38	212
2	82.09	366	28.49	212
3	69.65	366	35.75	212
4	72.39	366	33.06	212
5	71.39	366	31.72	212
Overall	74.18 ± 4.89	366.0 ± 0.0	31.88 ± 2.75	212.0 ± 0.0

Table 8: Complete repeated-run scores and optimization rollout counts (continued).

	ALFWorld		LiveMath	
Run	Score (%)	Num. of rollouts	Score (%)	Num. of rollouts
SkillOpt Parallel×7				
1	87.31	2,562	34.41	2,450
2	74.63	2,562	33.87	2,450
3	80.60	2,562	32.53	2,450
4	78.86	2,562	33.87	2,450
5	78.61	2,562	33.87	2,450
Overall	80.00 ± 4.64	$2{,}562.0 \pm 0.0$	33.71 ± 0.70	$2{,}450.0 \pm 0.0$
Skill-Forest $(5, 3)$				
1	78.61	1,895	34.14	1,599
2	76.87	1,613	32.53	1,015
3	75.87	1,910	36.02	2,273
4	84.08	1,700	33.60	464
5	83.83	2,084	33.06	1,779
Overall	79.85 ± 3.87	$1{,}840.4 \pm 186.1$	33.87 ± 1.34	$1{,}426.0 \pm 701.1$
Skill-Forest $(7, 2)$				
1	86.32	1,791	34.95	1,020
2	79.35	1,091	30.11	1,461
3	79.60	1,638	31.72	972
4	80.85	1,057	33.60	1,536
5	83.08	1,093	32.53	1,818
Overall	81.84 ± 2.91	$1{,}334.0 \pm 351.8$	32.58 ± 1.84	$1{,}361.4 \pm 359.5$
Skill-Forest $(7, 3)$				
1	80.10	2,052	36.02	2,334
2	82.34	2,100	35.22	2,250
3	85.32	1,924	38.44	2,314
4	78.11	1,725	36.56	1,777
5	85.07	2,068	33.06	2,336
Overall	82.19 ± 3.13	$1{,}973.8 \pm 154.4$	35.86 ± 1.96	$2{,}202.2 \pm 240.2$
Skill-Forest $(7, 3)$, shared split				
1	83.83	1,684	30.65	2,329
2	81.09	1,832	33.33	1,067
3	81.59	2,171	33.33	2,313
4	81.84	2,175	34.14	2,327
5	81.34	1,041	31.72	685
Overall	81.94 ± 1.09	$1{,}780.6 \pm 465.5$	32.63 ± 1.42	$1{,}744.2 \pm 804.0$

C.4 An ALFWorld Case Study: Shared Rule Refinements across Tree-local Winners

We examine the tree-local winners from one ALFWorld Forest experiment with seven trees. Trees 1, 2, 4, 5, and 6 produced eligible winners, whereas Trees 3 and 7 did not. The shared initial Skill already included core rules for systematic exploration, immediate pickup, transformation before placement, and direct delivery. We therefore focus on evolved refinements to these principles. The excerpts in Table 9 illustrate recurring search-discipline refinements alongside differences in operational emphasis.

The winners repeatedly refined the same search discipline: explicitly track visited or exhausted locations and avoid returning to locations that have already been checked. They also reinforced the transition from searching to execution once the target object is acquired, followed by direct delivery after any required transformation. Their differences primarily concern search priorities and task-specific handling. For example, Tree 2 favors an early fridge inspection for cooling tasks,

whereas Tree 6 emphasizes finding the object on ordinary surfaces before visiting the appliance. These differences show that shared operational principles can coexist with distinct, and occasionally conflicting, heuristics.

Select chose Tree 2, which achieved the highest full-training score of 84.62% and the highest candidate test score of 82.34%, matching Oracle in this construction. Merge achieved 82.84%, exceeding Select by 0.50 pp. Tree 5 achieved 100.00% on its own local validation subset but only 76.92% on the complete training split and 78.86% on the test split, illustrating why local validation scores alone do not establish which winner generalizes best and showing that reliance on a single validation split can introduce severe selection bias. This case is consistent with the possibility that different data perspectives produce similar core rule refinements, allowing a single well-selected winner to retain much of the candidate pool's utility.

Table 9: ALFWorld tree-local winner rule excerpts and scores with seven trees. Excerpts are illustrative rather than an exhaustive rule inventory. Bold identifies the Select winner. Full-train scores use the same complete training split; test scores are means over three held-out evaluations.

Tree	Shared search-discipline refinement	Differing operational emphasis	Full-train (%)	Test (%)
1	"Keep a strict internal visited set for the current episode."	When the required appliance or lamp is nearby, "use/open it as soon as the needed object is in hand".	82.05	79.60
2	**"mark it exhausted and do not return to it unless a new clue appears."**	**For cooling tasks, "inspect the fridge early even before exhaustive container search".**	**84.62**	**82.34**
3	No eligible local winner.	–	–	–
4	"Maintain a global visited set for the entire episode."	"bias the first search pass toward the most likely room context and nearby cluster for the object type".	82.05	81.84
5	"Search each location at most once per task unless new evidence appears."	"If placement fails, do not re-transform immediately".	76.92	78.86
6	"Maintain a strict visited set of exact locations"	For heating or cooling tasks, "Do not assume the goal item is already inside the microwave or fridge."	71.79	80.10
7	No eligible local winner.	–	–	–

C.5 An OfficeQA Case Study: Shared and Differing Rules across Tree-local Winners

We examine the seven-tree OfficeQA Forest used in Table 2. Six trees produced eligible winners; Tree 7 produced none. The initial Skill provides general retrieval, evidence-checking, and answer-verification instructions. The winners repeatedly refine exact source matching and complete-operand checks, while differing in the specific document and calculation cases they emphasize. Table 10 pairs these shared refinements with representative differences. The corresponding instructions also appear in the final merged Skill.

The winners share the need to match evidence precisely and assemble the complete data before calculation. Their distinct emphases cover trimmed statistics, note-based event lookup, ratio invariance, bulletin publication dates, exchange-rate direction, and index-based retrieval. These examples show variation within an overlapping rule set rather than disjoint strategies. The final Merge Skill scores 55.23% on the held-out test set, compared with 53.88% for Select. This difference is consistent with useful complementary refinements in the merged Skill, but does not isolate the contribution of any individual rule.

Table 10: OfficeQA tree-local winner rule summaries and scores with seven trees. Summaries include shortened excerpts and are not an exhaustive rule inventory; differing emphases need not be unique to one tree. Full-train scores use the same complete training split; test scores are means over three held-out evaluations. Trees 1 and 5 tie on full training; Tree 1 wins the novelty tie-break (0.276 versus 0.236 relative to the initial Skill). Bold Full-train and Test values identify the Select and diagnostic Oracle winners, respectively.

Tree	Shared evidence or computation refinement	Differing operational emphasis	Full-train (%)	Test (%)
1	"Never substitute a neighboring month, year, subtotal, or total row" for the requested period.	For a trimmed mean, "trim equally from both ends on transformed values".	**60.00**	53.88
2	List all operands and verify their dates before calculating.	Use a table note to locate the year an item was removed or retired.	48.00	43.41
3	"match the exact labels from the question" before extracting a table value.	A common currency or inflation factor cancels in a ratio.	52.00	38.57
4	For charts and extrema, "inspect the figure itself and count directly".	For month-end snapshots, the reporting issue may be published later; match the target row, not the cover date.	54.00	54.84
5	"Do not compute until every requested term or endpoint is present."	For currency conversion, "verify the exchange-rate quotation direction" before computing.	60.00	**55.04**
6	For a span across a boundary, "check adjacent bulletin issues for the missing boundary rows".	If literal wording fails, search the table or figure code or the contents entry, then inspect the cited page.	58.00	54.26
7	No eligible local winner.	–	–	–

D COMPLETE SKILL-FOREST ALGORITHM

Algorithm 1 Complete Pseudocode of Skill-Forest

Require: $\mathcal{D}_{\text{train}}, S_0, K, \rho, M, \delta, \mathbf{B}, A \in \{\text{Select}, \text{Merge}\}$; for Merge, $\tau = 0.15$ and G
Ensure: S_{forest}

Phase 1: Sample tree-local data and initialize Roots.
for $k = 1, \ldots, K$ **do**
  sample $(\mathcal{T}_k, \mathcal{V}_k)$ from $\mathcal{D}_{\text{train}}$ with training ratio ρ
  initialize tree k with Root Skill S_0
end for
Phase 2: Share Root validation across trees.
$\mathcal{V}_\cup \leftarrow \bigcup_{k=1}^{K} \mathcal{V}_k$
$\mathcal{R}_0 \leftarrow \text{Rollout}(S_0; \mathcal{V}_\cup)$ ▷ one outcome per unique task
for $k = 1, \ldots, K$ **do**
  $Q_{M_{\text{tgt}}}(S_0; \mathcal{V}_k) \leftarrow \text{Score}(\mathcal{R}_0|_{\mathcal{V}_k})$
end for
Phase 3: Grow trees by validation gain.
while unexpanded Roots remain or ($\mathbf{B}$ remains and queued nodes exist) **do**
  choose v: an unexpanded Root first, otherwise the highest-gain queued node
  let k be the tree containing v
  $\mathcal{R}_v \leftarrow \text{Rollout}(S_v; \mathcal{T}_k)$
  $\mathcal{P}_v \leftarrow \text{ProposePatches}(\mathcal{R}_v, M)$
  $n \leftarrow |\mathcal{V}_k|$
  $c_v \leftarrow \sum_{x \in \mathcal{V}_k} Q_{M_{\text{tgt}}}(S_v; x)$
  order $\mathcal{V}_k$ as $x_1, \ldots, x_n$, with tasks failed by v first
  for all $p \in \mathcal{P}_v$ **do**
    $S_u \leftarrow \text{Apply}(S_v, p)$
    $c_0 \leftarrow 0$
    for $j = 1, \ldots, n$ **do**
      $c_j \leftarrow c_{j-1} + Q_{M_{\text{tgt}}}(S_u; x_j)$
      $U_j \leftarrow (c_j + n - j - c_v)/n$
      if $U_j \le \delta$ **then**
        reject u; **break** ▷ early rejection
      end if
    end for
    if u was rejected **then**
      continue
    end if
    $g_u \leftarrow (c_n - c_v)/n$
    if $g_u > \delta$ **then**
      accept u as a child of v
      if u can be expanded within the depth budget **then**
        queue u with priority g_u
      end if
    end if
  end for
end while
Phase 4: Select and score one winner per eligible tree.
$\mathcal{W} \leftarrow [\,]$
for $k = 1, \ldots, K$ **do**
  $\mathcal{C}_k \leftarrow$ accepted non-Root nodes of tree k
  if $\mathcal{C}_k = \emptyset$ **then**
    continue
  end if
  $S_k^* \leftarrow \arg\max_{v \in \mathcal{C}_k} Q_{M_{\text{tgt}}}(S_v; \mathcal{V}_k)$
  $a_k \leftarrow Q_{M_{\text{tgt}}}(S_k^*; \mathcal{D}_{\text{train}})$
  add (S_k^*, a_k) to $\mathcal{W}$
end for

Algorithm 1 Complete Pseudocode of Skill-Forest (continued)

require $\mathcal{W} \neq [\,]$ ▷ no Root substitute
Phase 5a: Select an existing Skill.
if $A = \text{Select}$ **then**
  return the highest-scoring Skill in $\mathcal{W}$, breaking ties by TF–IDF novelty from S_0
end if
Phase 5b: Merge compatible rules across winners.
$a_{\max} \leftarrow \max_{(S,a)\in\mathcal{W}} a$
$\mathcal{H} \leftarrow [(S, a) \in \mathcal{W} : a \geq a_{\max} - \tau]$
arrange $\mathcal{H}$ in a fixed order
while $|\mathcal{H}| > 1$ **do**
  $\mathcal{H}_{\text{next}} \leftarrow [\,]$
  for adjacent pairs $(S_L, q_L), (S_R, q_R)$ in $\mathcal{H}$ **do**
    $U \leftarrow (S_L, q_L, S_R, q_R, \text{UnifiedDiff}(S_L, S_R))$
    $\widehat{S} \leftarrow G(U)$ ▷ one complete Skill
    $\widehat{q} \leftarrow Q_{M_{\text{tgt}}}(\widehat{S}; \mathcal{D}_{\text{train}})$
    add $(\widehat{S}, \widehat{q})$ to $\mathcal{H}_{\text{next}}$ ▷ no score-based rejection
  end for
  if $|\mathcal{H}|$ is odd **then**
    carry the unpaired item and its score to $\mathcal{H}_{\text{next}}$
  end if
  $\mathcal{H} \leftarrow \mathcal{H}_{\text{next}}$
end while
return the remaining Skill in $\mathcal{H}$