

# SkillCombiner: Optimizing Agent Skills via Edit Combination Search

Seungbeom Park

KAIST

kw\_park@kaist.ac.kr

Jaewon Chu

Korea University

allonsy07@korea.ac.kr

Hyunwoo J. Kim\*

KAIST

hyunwoo.jkim@kaist.ac.kr

## Abstract

Large language model agents are often guided by reusable natural-language skills that provide task-specific instructions. Existing work automatically refines these skills by using LLMs to propose and repeatedly incorporate textual edits from agent feedback. We find that the resulting performance depends substantially on which proposed edits are combined, and adding an edit can even degrade the skill. We therefore introduce SkillCombiner, which generates candidate edits and explicitly searches over their combinations. Across various settings, SkillCombiner matches or exceeds a strong baseline in mean performance while reducing optimizer cost by 75.2% and total optimization-stage cost by 21.5% on average. These results show that edit combination is an important optimization target in agent skill refinement.

## 1 Introduction

Reusable natural-language skills provide LLM agents with persistent guidance over strategies, behavioral constraints, and task-specific knowledge. Improving this guidance has become an important form of textual optimization for agent performance. Existing textual optimization methods use rollout feedback to revise natural-language components (Yuksekgonul et al., 2024; Agrawal et al., 2025). Recent skill optimization methods use LLMs to propose textual edits from rollout feedback and iteratively refine the skill document (Ni et al., 2026; Chen et al., 2026b; Zhang et al., 2026; Shen et al., 2026).

Existing skill optimization methods typically focus on generating better edits, while the choice of how to combine the proposed edits is often handled by accepting them all or delegating their integration to an LLM. As shown in Figure 1, however, proposed edits are not uniformly beneficial, and the resulting performance depends on which edits

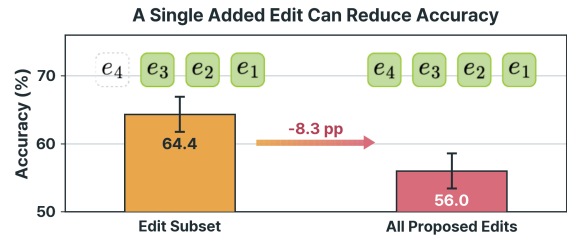


Figure 1: Mean validation accuracy ( $\pm$  standard error over 12 draws) for two combinations that differ by exactly one edit. Adding the edit lowers the mean by 8.3 percentage points ( $p = 0.047$ ; Appendix A).

are combined. We therefore separate edit proposal from combination selection, treating the latter as an explicit optimization problem for identifying an effective subset of proposed edits. This formulation is related to prior prompt optimization work that searches over components distributed across multiple LLM modules (Opsahl-Ong et al., 2024), but our setting instead considers combinations among multiple edits applied to a single skill document.

To optimize this choice directly, we introduce SkillCombiner. Within each optimization step, SkillCombiner constructs a fixed pool of edits and searches over their combinations. Across six model–benchmark settings, SkillCombiner matches or exceeds a strong baseline in mean performance. In the three Ministral-target settings used for cost analysis, it reduces optimizer-token usage by 75.2% and total token usage by 21.5% on average. Together with our observations, these results support treating edit combination as an explicit optimization target in agent skill refinement.

## 2 Related Work

**Agent skill optimization.** Agent skill optimization refines reusable natural-language skills from execution trajectories. SkillOpt alternates fast skill updates with slower meta-level updates under vali-

\* Corresponding author.

dation feedback (Yang et al., 2026b). Trace2Skill extracts reflections from individual trajectories and merges recurring lessons (Ni et al., 2026). Recent methods improve other parts of this pipeline. SkillCAT validates patches on the tasks that produced them, SkillComposer learns to create, refine, and merge skills, and SkillOpt-Lite allows a general-purpose coding agent to modify the agent harness (Chen et al., 2026b; Zhang et al., 2026; Shen et al., 2026). These methods mainly focus on generating, validating, or merging edits. We instead search over combinations of a fixed set of proposed edits.

**Optimization of textual components** TextGrad propagates natural-language feedback through an LLM system in a backpropagation-like manner, while GEPA evolves multiple prompt candidates through validation and LLM-based merging (Yuksekgonul et al., 2024; Agrawal et al., 2025). Other methods structure the edit space or route optimization signals within multi-component systems (Liu et al., 2026; Koh et al., 2026; Huang et al., 2026). MIPRO generates instruction and demonstration candidates, then uses Bayesian optimization to select their configuration across a multi-stage LLM program (Opsahl-Ong et al., 2024). Concurrent work also examines interactions among updates. MAA evaluates whether an update remains useful across batches, while LASKO studies how patch order affects the merged result (Yang et al., 2026a; Mahadevan, 2026). Our setting instead concerns multiple edits to the same skill document, whose effects may change when combined.

### 3 Method

#### 3.1 Problem Setup

We consider the common skill optimization setting in which a fixed agent is guided by a single natural-language skill document (Yang et al., 2026b; Ni et al., 2026; Chen et al., 2026b; Zhang et al., 2026). Given training tasks  $\mathcal{D}_{\text{train}}$ , an optimizer observes agent rollouts and revises the initial skill  $s_i$ . The agent model, tools, and execution framework remain fixed. The objective is to find a revised skill that maximizes held-out validation performance:

$$s^* = \arg \max_{s \in \mathcal{S}} \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{z \in \mathcal{D}_{\text{val}}} R(A(z; s)), \quad (1)$$

where  $A(z; s)$  denotes the agent’s rollout on task  $z$  when guided by skill  $s$ , and  $R$  denotes the task reward.

#### 3.2 SkillCombiner

**SkillCombiner** separates edit proposal from combination selection. As shown in Figure 2, it first constructs a fixed pool of ranked candidate edits from training rollouts. It then searches over budget-constrained combinations that maximize held-out validation performance.

**Rollout and Edit Proposal.** Given an initial skill  $s_0$ , the fixed target agent executes the training tasks to produce trajectories, which are separated into successful and failed cases and partitioned into  $m$  minibatches  $\{\mathcal{B}_i\}_{i=1}^m$ , each containing at most  $b$  trajectories. For each minibatch  $\mathcal{B}_i$ , the optimizer receives  $s_0$  and the corresponding rollout feedback and proposes up to  $L$  candidate edits in priority order:

$$e_{i,1} \succ e_{i,2} \succ \dots \succ e_{i,L_i}, \quad L_i \leq L, \quad (2)$$

where  $e_{i,1}$  is expected to have the greatest performance improvement. This proposal stage produces a fixed candidate pool, which is then mapped to a discrete coordinate space for combination search.

**Combination Search.** Each candidate combination is represented by an integer vector

$$\mathbf{x} = (x_1, \dots, x_m), \quad 0 \leq x_i \leq L_i \quad (3)$$

Given a total edit budget  $L_{\text{tot}}$ , feasible combinations satisfy

$$\sum_{i=1}^m x_i \leq L_{\text{tot}}. \quad (4)$$

Each coordinate  $x_i$  specifies the number of top-ranked edits selected from  $\mathcal{B}_i$ . For example,  $x_i = 2$  selects  $e_{i,1}$  and  $e_{i,2}$ . This parameterization turns candidate-edit selection into a bounded integer optimization problem.

For each feasible  $\mathbf{x}$ , the selected edits are applied to  $s_0$ , and the resulting skill is evaluated on held-out validation tasks. We search this discrete space using an integer-grid variant of Nelder–Mead, which proposes new integer combinations based on the validation scores of previously evaluated points. Further details are provided in Appendix B. The search evaluates at most  $Q$  unique combinations and terminates early if the best validation score does not improve for  $p$  consecutive search steps.

**Reselection and Final Skill.** To mitigate noise in individual validation results, the top  $k$  combinations are re-evaluated over  $r$  additional runs, and

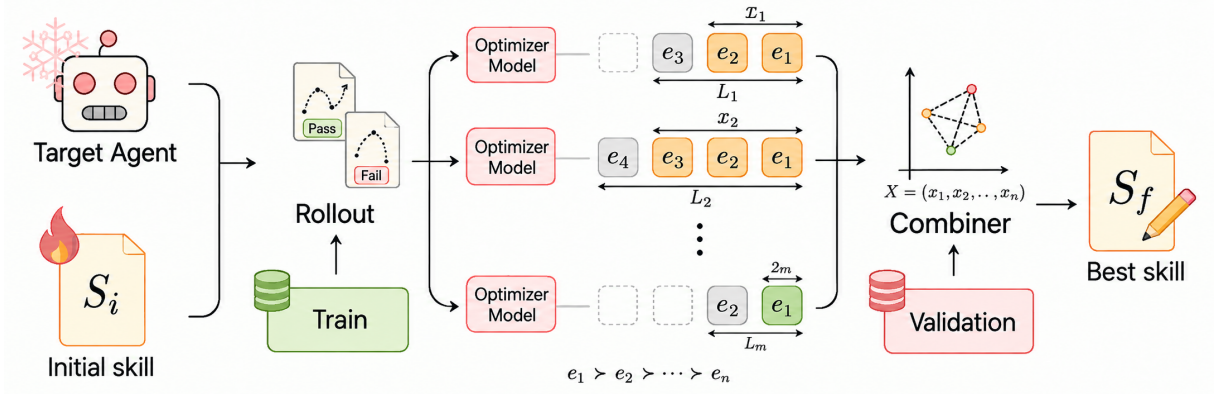


Figure 2: Overview of SkillCombiner. Candidate edits are proposed from minibatch-level rollout feedback and ordered by priority. SkillCombiner then searches over budget-constrained combinations, re-evaluates the most promising candidates, and applies the selected edits without an additional LLM-based merge.

the combination with the highest mean score is selected. Its edits are applied to  $s_0$  deterministically in minibatch order, without LLM-based merging. The resulting skill is adopted only if it improves over the current skill and is returned as  $s_f$ .

## 4 Experiments

### 4.1 Experimental Setup

SkillCombiner is compared with SkillOpt, a strong iterative baseline that optimizes the same type of reusable skill document from rollout feedback (Yang et al., 2026b). We take the benchmark-specific reflection prompt from the released configuration of the baseline and only add a single prompt to sort the edits by rank. The added prompt is in Appendix D.

Experiments cover LiveMath, Spreadsheet-Bench, and ALFWorld under two target models, Qwen3.5-9B and Ministral3-14B, representing mathematical reasoning, spreadsheet manipulation, and multi-step interaction, respectively (He et al., 2026; Ma et al., 2024; Shridhar et al., 2020; Qwen Team, 2026; Mistral AI, 2025). Each setting is evaluated over three optimization seeds. We set SkillCombiner’s within-step total edit-budget upper bound to  $L_{\text{tot}} = 14$ . SkillCombiner is run for one optimization epoch, whereas the baseline follows its released configuration.

### 4.2 Overall Performance

Table 1 reports test accuracy across two target models and three benchmarks, averaged over three optimization seeds. SkillCombiner remains competitive with SkillOpt across all six settings, achieving equal or higher mean accuracy in each case. This

result suggests that a single epoch edit-proposal followed by explicit combination selection can achieve performance comparable to iterative skill optimization. ALFWorld with Ministral3-14B shows the weakest response to skill optimization, where neither method improves over the no-skill baseline.

### 4.3 Optimization Efficiency

Rather than repeating edit proposal and meta updates each round, SkillCombiner proposes candidates once and searches over their combinations, which requires substantially fewer optimizer calls. As shown in Table 2, it reduces optimizer-token usage by 75.2% on average, while reducing total token usage by 21.5%. It also achieves a  $1.84\times$  speedup in end-to-end optimization wall-clock time on average (Table 3). The smaller reduction in total tokens reflects the shared cost of training and validation rollouts. SkillOpt repeatedly collects training rollouts across optimization rounds, whereas SkillCombiner uses validation rollouts to compare a fixed set of proposed edits.

The reduction advantage is therefore smaller on rollout-intensive benchmarks such as ALFWorld, where target-agent execution dominates the total cost. In contrast, the reduction becomes more valuable when the optimizer is a stronger or more expensive model, since SkillCombiner requires far fewer optimizer calls. Appendix C further shows that similar performance can be retained with fewer validation examples, demonstrating its strong potential as a more efficient method.

| Model          | Method      | LiveMath                           | Spreadsheet                        | ALFWorld                           |
|----------------|-------------|------------------------------------|------------------------------------|------------------------------------|
| Qwen3.5-9B     | No Skill    | 25.00                              | 26.79                              | 59.38                              |
|                | Base Skill  | 22.58                              | 23.21                              | 81.25                              |
|                | SkillOpt    | 50.54 $\pm$ 4.06                   | <b>28.33 <math>\pm</math> 1.21</b> | 85.42 $\pm$ 4.78                   |
|                | <b>Ours</b> | <b>52.15 <math>\pm</math> 4.66</b> | <b>28.33 <math>\pm</math> 0.45</b> | <b>88.54 <math>\pm</math> 4.77</b> |
| Ministral3-14B | No Skill    | 29.03                              | 19.29                              | <b>59.38</b>                       |
|                | Base Skill  | 24.19                              | 24.64                              | <b>59.38</b>                       |
|                | SkillOpt    | 56.99 $\pm$ 1.00                   | 25.21 $\pm$ 2.09                   | 58.33 $\pm$ 1.81                   |
|                | <b>Ours</b> | <b>59.41 <math>\pm</math> 3.25</b> | <b>30.71 <math>\pm</math> 4.56</b> | <b>59.38 <math>\pm</math> 6.21</b> |

Table 1: Test accuracy (%), reported as mean  $\pm$  standard deviation over three optimization seeds. Best means are bold. ALFWorld uses the same fixed 32-example test subset across methods and seeds. Its Qwen3.5-9B setting uses GPT-5.4 mini as the optimizer for both methods while all other settings use Qwen3.5-9B.

| Benchmark      | Total tokens (M)      | Total reduction           | Optimizer reduction       |
|----------------|-----------------------|---------------------------|---------------------------|
| LiveMath       | <b>1.318</b> /1.846   | $\downarrow$ 28.6%        | $\downarrow$ 82.2%        |
| Spreadsheet    | <b>10.154</b> /15.612 | $\downarrow$ 35.0%        | $\downarrow$ 72.9%        |
| ALFWorld       | <b>25.481</b> /25.750 | $\downarrow$ 1.0%         | $\downarrow$ 70.3%        |
| <b>Average</b> | —                     | $\downarrow$ <b>21.5%</b> | $\downarrow$ <b>75.2%</b> |

Table 2: Optimization cost with Ministral3-14B as the target model, averaged over three optimization seeds. Total token usage is reported as SkillCombiner/SkillOpt in millions. Reductions are measured relative to SkillOpt and averaged across benchmarks.

| Benchmark      | SkillOpt         | SkillCombiner                      | Speedup                        |
|----------------|------------------|------------------------------------|--------------------------------|
| LiveMath       | 36.0 $\pm$ 8.0   | <b>35.0 <math>\pm</math> 4.5</b>   | 1.03 $\times$                  |
| Spreadsheet    | 466.5 $\pm$ 18.8 | <b>146.4 <math>\pm</math> 8.9</b>  | 3.19 $\times$                  |
| ALFWorld       | 698.2 $\pm$ 72.7 | <b>539.2 <math>\pm</math> 74.5</b> | 1.29 $\times$                  |
| <b>Average</b> | —                | —                                  | <b>1.84<math>\times</math></b> |

Table 3: Optimization wall-clock time with Ministral3-14B as the target model, averaged over three optimization seeds. Times are reported in minutes as mean  $\pm$  standard deviation. Speedups are measured relative to SkillOpt and averaged across benchmarks.

#### 4.4 Effect of Optimization Epochs

The baseline is run for multiple optimization epochs, whereas SkillCombiner is run for only one epoch. Figure 3 compares each method across epochs. SkillOpt does not converge within a single epoch, and its accuracy increases across epochs without exceeding SkillCombiner. Although SkillCombiner has a higher cost per epoch, a single epoch matches or exceeds SkillOpt across all of its epochs, and the total cost remains lower.

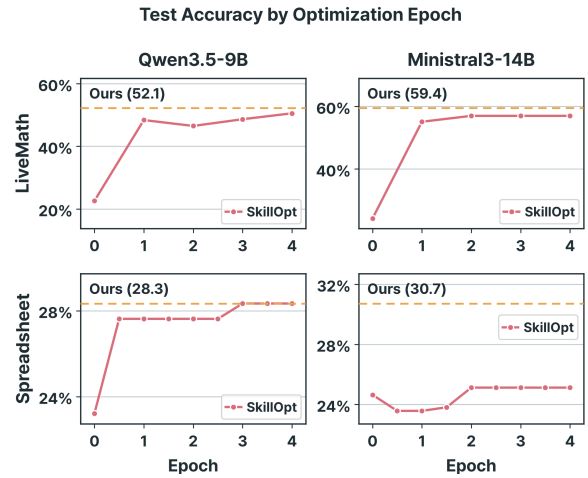


Figure 3: Test accuracy of SkillOpt across optimization epochs on LiveMath and Spreadsheet. Epoch 0 is the initial skill document. The dashed line marks SkillCombiner’s performance. Results are averaged over three optimization seeds.

#### 4.5 Cross-Model Transfer

To examine whether SkillCombiner search produces source-model-specific skills, skills optimized with Ministral3-14B are evaluated on Qwen3.5-9B without further optimization. As shown in Table 4, SkillCombiner matches or exceeds the mean performance of SkillOpt across all three benchmarks. This result suggests that SkillCombiner remains effective after transfer to a different model.

#### 4.6 Search Strategy Ablation

Figure 4 compares random search, hill climbing (Russell and Norvig, 2020), forward selection (Kohavi and John, 1997), and the discrete Nelder-Mead variant used in SkillCombiner. All search strategies outperform applying the proposed edits without search. Forward selection and the discrete

| Benchmark   | Ours                               | SkillOpt                            | $\Delta$ |
|-------------|------------------------------------|-------------------------------------|----------|
| LiveMath    | <b>59.41 <math>\pm</math> 5.49</b> | 52.69 $\pm$ 8.47                    | +6.72    |
| Spreadsheet | <b>29.17 <math>\pm</math> 1.69</b> | 26.55 $\pm$ 0.90                    | +2.62    |
| ALFWorld    | <b>80.21 <math>\pm</math> 7.86</b> | <b>80.21 <math>\pm</math> 10.05</b> | 0.00     |

Table 4: Cross-model transfer from Ministral3-14B to Qwen3.5-9B. Test accuracy (%) is reported as mean  $\pm$  standard deviation.  $\Delta$  denotes SkillCombiner minus SkillOpt in percentage points.

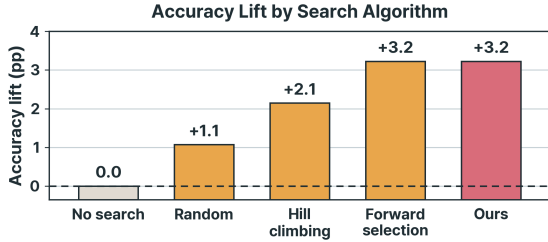


Figure 4: Test accuracy comparison across search methods on LiveMath. All methods use the same validation budget, and results are averaged over three optimization seeds.

Nelder–Mead variant achieve the largest gains, while even random search improves test performance. These results support the importance of explicitly searching over candidate-edit combinations.

#### 4.7 Case Study

The effect of applying several edits together is not determined by the edits themselves. Figure 5 shows an example of the interaction between the proposed edits. The first adds local guidelines for writing formulas in cells. The second adds execution policies for iteration, row deletion, and None handling. Each improves test accuracy over the initial skill by 1.97% and 2.50% while improving disjoint sets of problems. Their instructions do not conflict and cover different parts of the task, which suggests that merging them would improve accuracy further. Applied together, however, the gain over the initial skill drops to 0.36%. When the two are combined, the guidelines of the first edit affect how the execution policies of the second are applied and offset its benefit. This reverts what the second edit had fixed and causes regressions on problems that were already correct. The optimal solution is to accept the second edit alone, which is what SkillCombiner selects. Edits can therefore interact in ways that their text does not reveal, making the choice of combination an optimization axis of its own.

(A) Edit 1 - local guidelines

### Writing Formulas to Cells

- Always write formulas as strings and assign directly to `cell.value`.
- Do not skip cells in the target range.
- Ensure all target cells are populated.

$e_2$

$e_1$

+1.97pp

(B) Edit 2 - execution policy

### Iteration Safety

- Always use `values_only=True` with `iter_rows()`.

### Row Deletion Safety

- Sort row indices in descending order.

### None Value Handling

- Check for None explicitly.

$e_2$

$e_1$

+2.50pp

(C) Combined edit

### Iteration Safety

...

### Writing Formulas to Cells

...

$e_2$

$e_1$

+0.36pp

Figure 5: Two edits proposed for the same initial skill on SpreadsheetBench with Ministral3-14B. (A) and (B) apply each edit alone, and (C) applies both. Numbers are test accuracy gains over the initial skill in percentage points.

## 5 Discussion

SkillCombiner is run for a single epoch in our experiments but can be extended to multiple epochs. Each additional epoch proposes new edits based on the updated skill, which may further improve performance.

Token usage and wall-clock time can be affected by different task- and system-level factors, and their savings may therefore differ across tasks. For example, wall-clock time may depend on the degree of parallelism available for rollouts, CPU-bound environment execution, and the sequential decoding length of the target agent. Therefore, the practical benefit may vary across tasks and system configurations.

## 6 Conclusion

We introduced SkillCombiner, a skill-optimization pipeline that separates edit proposal from combination search. SkillCombiner explicitly searches over combinations of proposed edits. Across various settings, it remains competitive with iterative skill optimization while reducing optimizer-side computation. These results suggest that how proposed edits are combined is an important part of skill refinement, motivating combination search as an explicit optimization layer.

## Limitations

**Combination-space parameterization.** To make combination search tractable, our formulation represents each minibatch by the number of top-ranked edits selected from it. This parameterization depends on the quality of the optimizer-generated ranking and does not cover arbitrary subsets of edits within a minibatch. It may therefore overlook useful combinations involving lower-ranked edits.

**Scalability.** Our experiments cover two target models and relatively small candidate pools. The performance and computational trade-offs of combination search may differ with stronger optimizer or target models, larger edit pools, and longer skill documents. In particular, validation cost may become more substantial as the search space grows.

**Search strategy.** We evaluate only a limited set of search algorithms, with the direct comparison conducted on LiveMath. While the results support explicitly searching over candidate-edit combinations, they do not establish discrete Nelder–Mead as the optimal strategy. The most effective search algorithm may depend on the candidate space, validation budget, and task.

## References

- Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziem, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. 2025. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *First Workshop on Foundations of Reasoning in Language Models*.
- Prasanna Balaprakash, Stefan M. Wild, and Paul D. Hovland. 2011. Can search algorithms save large-scale automatic performance tuning? *Procedia Computer Science*, 4:2136–2145.
- Xiaoyi Bao, Yuanzhen Xie, Yunzhi Tan, Jinghang Gu, Zhongqing Wang, Chu-Ren Huang, Bo Hu, and Zang Li. 2026. Skillmentor: Llm agent self-evolution via learning blind-spot diagnosis. *Preprint*, arXiv:2607.27360.
- Jiangwang Chen, Zixin Song, Junlin Liu, Shuaiyu Zhou, Haiyan Wu, Haihan Shi, Chenxi Zhou, Hanqing Li, Xiao Yang, Da Zhu, Guanjin Jiang, Hai Wan, and Xibin Zhao. 2026a. Decoevo: Score-decoupled co-evolution of solver and rubric-generator skills in text space. *Preprint*, arXiv:2607.25675.
- Kunfeng Chen, Qihuang Zhong, Juhua Liu, and Bo Du. 2026b. Skillcat: Contrastive assessment and topology-aware skill self-evolution for llm agents. *arXiv preprint arXiv:2606.13317*.
- Linyang He, Qiyao Yu, Hanze Dong, Baohao Liao, Xinxing Xu, Micah Goldblum, Jiang Bian, and Nima Mesgarani. 2026. Livemathematicianbench: A live benchmark for mathematician-level reasoning with proof sketches. *arXiv preprint arXiv:2604.01754*.
- Suizhi Huang, Mei Li, Han Yu, and Xiaoxiao Li. 2026. Textresnet: Decoupling and routing optimization signals in compound ai systems via deep residual tuning. *arXiv preprint arXiv:2602.08306*.
- Changguo Jia, Tianqi Zhao, Zhiyou Xiao, Weiming Zhang, and Minghui Zhou. 2026. Veriskill: A self-evolution framework for program verification skills. *Preprint*, arXiv:2607.27733.
- Derek Koh, Jinghui Mo, Benjamin H Le, Jiening Zhan, Baofen Zheng, Kevin Bevis, Nathaniel C Owen, Lauren Elizabeth Charney, Wenqiong Liu, and Jingwei Wu. 2026. Contrastive reflection for iterative prompt optimization. *arXiv preprint arXiv:2606.30840*.
- Ron Kohavi and George H. John. 1997. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1–2):273–324.
- Haoyue Liu, Zhichao Wang, Yongxin Guo, Hao-ran Shou, and Xiaoying Tang. 2026. Adaptive prompt structure factorization: A framework for self-discovering and optimizing compositional prompt programs. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11690–11714.
- Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. 2024. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908.
- Sridhar Mahadevan. 2026. Agentic skill optimization over lie algebroids. *arXiv preprint arXiv:2607.11493*.
- Mistral AI. 2025. Ministral-3-14B-Instruct-2512. <https://huggingface.co/mistralai/Ministral-3-14B-Instruct-2512>. Model card.
- Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjin Jiang. 2026. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*.
- Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366.

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. 2026. [Self-supervised skill optimization](#). *Preprint*, arXiv:2607.28777.

Qwen Team. 2026. Qwen3.5-9B. <https://huggingface.co/Qwen/Qwen3.5-9B>. Model card.

Stuart Russell and Peter Norvig. 2020. *Artificial Intelligence: A Modern Approach*, 4th edition. Pearson.

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

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

Mingyu Yang, Keye Zheng, Congchao Cheng, Yujie Liu, Xingkang Lu, Fan Jiang, and Yefei Zheng. 2026a. Marginal advantage accumulation for memory-driven agent self-evolution. *arXiv preprint arXiv:2606.20475*.

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

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

Qi Zhang, Zhaopeng Feng, Xiaonan Shi, Xiaomeng Hu, Chu Liu, Pengjun Xie, Xiaobin Wang, Jieping Ye, Bryan Hooi, Haobo Wang, et al. 2026. Skill-composer: Learning to evolve agent skills for specification and generalization. *arXiv preprint arXiv:2606.06079*.

ChaoJin Zhao and Xuan Jiang. 2026. [Verifiable self-evolution for open-ended dialogue skills via future-feedback prediction](#). *Preprint*, arXiv:2607.18973.

## A Statistical Verification

### A.1 Combination effect

Let  $\mu_i$  denote the expected validation score of combination  $i$ . We test

$$H_0 : \mu_1 = \mu_2 = \dots = \mu_{32},$$

$$H_1 : \exists i \neq j \text{ such that } \mu_i \neq \mu_j.$$

Each combination is evaluated three times. We use the ratio of between-combination variation to within-combination variation as the test statistic. A permutation test rejects  $H_0$  ( $p < 0.001$ ), showing that combination choice affects validation performance beyond repeated-evaluation noise.

### A.2 Harmful edit effect

For the example in Figure 1, let  $\mu_{-e}$  and  $\mu_{+e}$  denote the expected validation scores without and with edit  $e$ , respectively. We test

$$H_0 : \mu_{-e} = \mu_{+e},$$

$$H_1 : \mu_{-e} \neq \mu_{+e}.$$

Each condition is evaluated 12 times. Their mean validation accuracies are 0.644 and 0.560, respectively. A two-sided permutation test rejects  $H_0$  at the 0.05 significance level ( $p = 0.047$ ). These results show that adding a proposed edit can lower validation performance.

## B Integer-Grid Nelder–Mead

We use the standard Nelder–Mead coefficients

$$(\alpha, \gamma, \rho, \sigma) = (1, 2, 0.5, 0.5)$$

for reflection, expansion, contraction, and shrink, respectively. Let  $\mathbf{x}_b$  and  $\mathbf{x}_w$  denote the best and worst simplex vertices, and let  $\mathbf{c}$  be the centroid excluding  $\mathbf{x}_w$ . The four operations are

$$\mathbf{x}_r = \mathbf{c} + \alpha(\mathbf{c} - \mathbf{x}_w),$$

$$\mathbf{x}_e = \mathbf{c} + \gamma(\mathbf{x}_r - \mathbf{c}),$$

$$\mathbf{x}_c = \mathbf{c} + \rho(\mathbf{x}_w - \mathbf{c}),$$

$$\mathbf{x}'_j = \mathbf{x}_b + \sigma(\mathbf{x}_j - \mathbf{x}_b).$$

We otherwise use the standard Nelder–Mead update rules.

Because these operations produce real-valued points, we map each proposal to a feasible integer coordinate (Balaprakash et al., 2011). For each component, we consider its clipped floor and ceiling values. We enumerate the coordinates formed from these values and remove those whose sum exceeds  $L_{\text{tot}}$ . Among the remaining coordinates, we choose the nearest one that has not yet been evaluated. If all of them have already been evaluated, we reuse the cached score of the nearest coordinate. Distance ties are resolved lexicographically.

If none of the neighboring coordinates satisfies the total edit budget, we round and clip each component. We then repeatedly subtract one from the first largest positive component until the coordinate sum is at most  $L_{\text{tot}}$ .

Integer projection can map several simplex proposals to the same coordinate. We therefore terminate a search start after three consecutive iterations with no new evaluation, or when all simplex

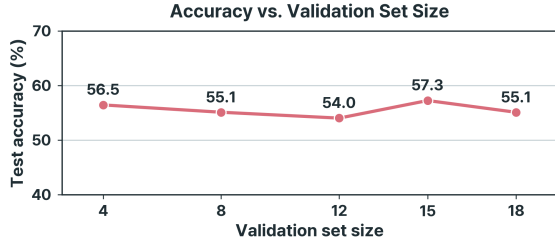


Figure 6: Effect of validation-set size on LiveMath test accuracy with Ministral3-14B. We report mean accuracy over three seeds. Performance remains within approximately one standard deviation of the result reported in Table 1.

vertices map to the same integer coordinate. The implementation supports random restarts, but all reported experiments use a single start ( $m_{\text{starts}} = 1$ ).

### C Effect of Validation Set Size

Because SkillCombiner makes relatively few optimizer calls, most of its optimization cost comes from validation probes, whose cost scales approximately with the number of validation examples. We therefore vary the validation-set size on LiveMath with Ministral3-14B and examine test accuracy.

As shown in Figure 6, test accuracy varies little as the validation set is reduced from 18 to 4 examples. This suggests that useful differences among candidate combinations can be detected with a small probe set, leaving room to further reduce the optimization cost of SkillCombiner.

## D Implementation Details

To facilitate a controlled comparison, we align our implementation and experimental setup with the released SkillOpt configuration wherever applicable (Yang et al., 2026b). We describe the remaining method-specific settings below.

### D.1 Optimizer Prompts

Both methods use the benchmark-specific reflection prompt from SkillOpt’s released configuration. The prompt asks the optimizer to identify recurring patterns across a rollout minibatch, respect the current edit allowance, and return a JSON patch with four edit operations (append, insert\_after, replace, and delete). It is populated with the same framework-constructed inputs, namely rollout trajectories, the current skill, and the current edit budget.

SkillCombiner appends the following instruction to this prompt. The resulting order defines the prefix candidates used in combination search.

The optimizer treats patch.edits emission order as priority when constructing prefix candidates; list edits from highest to lowest expected validation impact.

### D.2 Target-Model Configuration

We use Qwen3.5-9B as the optimizer model. During skill optimization and validation, target-model responses are sampled with temperature 0.7. Final test and transfer evaluations use temperature 0 with a single generation per example. The optimizer uses temperature 0 with thinking disabled. The maximum generation lengths are 16,384 tokens for the target model and 64,000 tokens for the optimizer. We fix the optimization seeds to 42, 44, and 45 for all experiments reporting three-seed results.

### D.3 Hyperparameters

We run SkillCombiner for one optimization epoch. Each optimization batch contains at most  $m = 6$  reflection minibatches, with at most  $b = 8$  rollout trajectories per minibatch. For each minibatch, the optimizer proposes at most  $L = 4$  edits. Most benchmarks use one optimization batch per epoch, whereas SpreadsheetBench uses two.

We use  $L_{\text{tot}} = 14$  as a fixed within-step upper bound across all settings, calibrated with reference to the early-step edit allowances of SkillOpt’s released SpreadsheetBench configuration. The combination search evaluates at most  $Q = 28$  unique candidates and stops early after  $p = 7$  probes without improvement. After search, we retain the top  $k = 4$  combinations and evaluate each with  $r = 2$  additional draws. The selected combination is adopted only when it improves on the current skill; score ties favor the shorter resulting skill.

For SkillOpt, we use its released four-epoch configuration and cosine edit-budget schedule. Unless otherwise specified, SkillCombiner follows the corresponding SkillOpt settings.

### D.4 Benchmarks

We evaluate on LiveMath, SpreadsheetBench, and ALFWorld. LiveMath evaluates mathematical reasoning on a collection of math problems. SpreadsheetBench evaluates an agent’s ability to inspect and modify spreadsheets through multi-step tool interactions. ALFWorld evaluates text-based embodied agents on household tasks that require planning and interaction with objects in an environment.

For LiveMath and SpreadsheetBench, we use the same task splits and benchmark configurations as SkillOpt. LiveMath contains 35 training, 18 validation, and 124 test examples, while SpreadsheetBench contains 280 test examples. For ALFWorld, we retain the SkillOpt training and validation splits, containing 39 and 18 examples, respectively, but evaluate on the first 32 examples of the test split. This fixed subset is used for all methods and optimization seeds.

### D.5 Optimizer Configuration for ALFWorld

For the Qwen3.5-9B target-model setting on ALFWorld, both SkillOpt and SkillCombiner use GPT-5.4 mini as the optimizer. The long rollout contexts could not be processed within the memory constraints of the available hardware, so this configuration was chosen for execution feasibility rather than based on comparative performance. We did not evaluate a Qwen3.5-9B optimizer for this setting. The optimizer is held fixed across methods within every setting.

## E Detailed Optimization Cost

Table 5 provides the detailed cost composition for each benchmark, including total token usage and optimizer-token usage, together with their reductions relative to SkillOpt.

## F Concurrent Work

Several studies on agent skill evolution were released after our experimental design and baseline suite had been established. Self-Supervised Skill Optimization learns from unlabeled tasks using comparative judgments and behavior-level evidence aggregation (Peng et al., 2026). DecoEvo jointly evolves solver and rubric-generator skills using decoupled optimization signals (Chen et al., 2026a). SkillMentor trains a mentor policy to discover executor blind spots and convert them into corrective skills (Bao et al., 2026). Verifiable Self-Evolution for Open-Ended Dialogue Skills constructs a future-feedback prediction objective from logged user responses, enabling verifiable optimization without regenerating user feedback (Zhao and Jiang, 2026). VeriSkill develops failure-attribution, lesson-abstraction, and executable-validation mechanisms for program-verification skills (Jia et al., 2026).

These studies address complementary problems, including learning without ground-truth rewards,

evaluator co-evolution, blind-spot discovery, verifiable dialogue feedback, and domain-specific failure attribution. Our work instead studies the joint selection of edits proposed for the same skill document. Because these methods were released during the final stage of our manuscript preparation and use different optimization objects or feedback regimes, we discuss them separately rather than include them as baselines.

| Benchmark                | Method      | Total tokens               |           | Optimizer tokens          |           |
|--------------------------|-------------|----------------------------|-----------|---------------------------|-----------|
|                          |             | Usage                      | Reduction | Usage                     | Reduction |
| LiveMath                 | SkillOpt    | $1.846 \pm 0.100\text{M}$  | –         | $0.670 \pm 0.038\text{M}$ | –         |
|                          | <b>Ours</b> | $1.318 \pm 0.099\text{M}$  | ↓ 28.6%   | $0.119 \pm 0.003\text{M}$ | ↓ 82.2%   |
| SpreadsheetBench         | SkillOpt    | $15.612 \pm 2.365\text{M}$ | –         | $0.998 \pm 0.247\text{M}$ | –         |
|                          | <b>Ours</b> | $10.154 \pm 0.767\text{M}$ | ↓ 35.0%   | $0.270 \pm 0.024\text{M}$ | ↓ 72.9%   |
| ALFWorld                 | SkillOpt    | $25.750 \pm 0.870\text{M}$ | –         | $0.836 \pm 0.061\text{M}$ | –         |
|                          | <b>Ours</b> | $25.481 \pm 1.869\text{M}$ | ↓ 1.0%    | $0.248 \pm 0.023\text{M}$ | ↓ 70.3%   |
| <b>Average reduction</b> |             | –                          | ↓ 21.5%   | –                         | ↓ 75.2%   |

Table 5: Full optimization-cost composition with Ministral3-14B as the target model. We report mean  $\pm$  standard deviation over three optimization seeds for total token usage and optimizer-token usage. Total tokens include both optimizer usage and target-agent tokens consumed by training and validation rollouts, while optimizer tokens cover all skill-update calls, including SkillOpt’s fast updates and slow meta updates. Reductions are computed relative to SkillOpt.