STRATUNE: ADAPTIVE SELECTION OF REVISION OPERATORS FOR SELF-EVOLVING LLM SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) can learn reusable textual skills from execution feedback without updating their parameters, but effectively deciding how to revise these skills remains a key challenge. Existing methods typically rely on a fixed revision operator, a search strategy and the revision forms applied under it. However, we observe that no single revision operator consistently performs best across tasks, and repeatedly applying an unsuitable operator can limit further improvement. We propose **StraTune** (**stra**tegy-guided skill **tun**ing), which lets a frozen optimizer LLM choose the revision operator at every round from the optimization state, which is defined as the current execution feedback together with the recorded outcomes of earlier strategies and forms. Candidate skills from every revision operator pass one candidate evaluation, which screens for gains and regressions on a small sample set and validates them on a larger one, and every outcome is written back to the optimization state for later choices. Across four benchmarks and two LLM settings, StraTune outperforms all five baselines in most settings. Ablations attribute the gains to the adaptive choice of the revision operator, since fixed, random, scheduled, and bandit strategy choices all score lower, and skills learned with a small target LLM also improve a stronger one.

1 INTRODUCTION

Large language models (LLMs) increasingly rely on *skills*, reusable text that tells a model how to perform a task, such as instructions, worked examples, and procedures, applied without changing the model's weights (Brown et al., 2020; Wei et al., 2022; Kojima et al., 2022; Liu et al., 2026). Skills have become a standard component of agent systems. An agent may load folders of instructions and scripts on demand (Zhang et al., 2025), call programs from a skill library (Wang et al., 2023), follow workflows induced for web navigation (Wang et al., 2024b; Zheng et al., 2025), consult a memory of insights that carries across reasoning problems (Suzgun et al., 2026), read a playbook of strategies for agent and finance tasks (Zhang et al., 2026), or carry domain knowledge into specialized fields (Mai et al., 2025). Skills were first written by hand: a prompt engineer encodes expert task knowledge into instructions and tunes them by trial and error. Because this effort must be repeated for every new task and model, it motivated automatic search over the instruction text (Zhou et al., 2022; Pryzant et al., 2023; Yang et al., 2024; Fernando et al., 2023; Guo et al., 2024; Wang et al., 2024a). More recently, **self-evolving skills** have become common. Here, the *target LLM*–the LLM that executes the task–receives execution feedback, and an *optimizer LLM* turns the feedback into revisions of the skill (Gao et al., 2025), whether as verbal reflections (Shinn et al., 2023; Madaan et al., 2023), insights distilled from trajectories (Zhao et al., 2024; Ouyang et al., 2026), code verified in the environment (Wang et al., 2023), or textual gradients and edits (Yuksekgonul et al., 2024; Agrawal et al., 2026; Yang et al., 2026; Ni et al., 2026).

Despite their variety, these methods share one loop. The target LLM rolls out the current skill on training tasks, a supervision signal (e.g., human judgments, an LLM verdict, or test cases) marks each outcome as success or failure, and the optimizer LLM revises the skill. They differ in two choices made in every optimization round, and Figure 1(a) compares these choices and lists the corresponding existing methods. The first is **how candidate search is organized** (columns in Figure 1(a)). TextGrad (Yuksekgonul et al., 2024) and SkillOpt (Yang et al., 2026) revise the current skill directly (I1); GEPA (Agrawal et al., 2026) and Voyager (Wang et al., 2023) refine a separate working version before it replaces the current skill (I2); OPRO (Yang et al., 2024), MIPROv2 (Opsahl-Ong et al., 2024), and Trace2Skill (Ni et al., 2026) generate several candidate

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

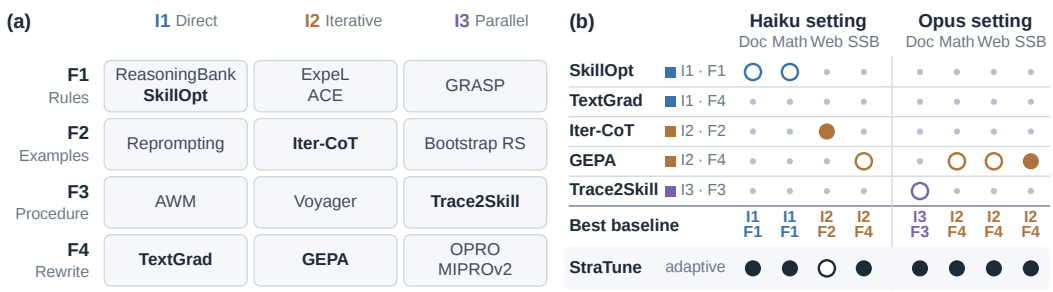


Figure 1: **(a)** Existing skill-learning methods by search strategy (columns) and revision form (rows); bold marks the baselines used in this paper. **(b)** Test scores (Table 1) on DocVQA (Doc), LiveMath (Math), Mind2Web (Web), and SpreadsheetBench (SSB) under two model settings; filled circles mark the best method in a column, open circles the second best. The row labelled *Best baseline* gives the strategy and form of the best baseline in each column, which change from column to column, whereas StraTune ranks first in most.

skills and rank or merge them (I3). The second is **what kind of change is written into the skill** (rows in Figure 1(a)). SkillOpt, ExpeL (Zhao et al., 2024), and GRASP (Moll et al., 2026) add conditional rules (F1); Iter-CoT (Sun et al., 2024), Reprompting (Xu et al., 2023), and bootstrapped demonstrations in DSPy (Khattab et al., 2023) add worked examples (F2); AWM (Wang et al., 2024b) and Voyager write reasoning procedures or workflows (F3); whereas TextGrad, GEPA, and OPRO rewrite the skill in full (F4). Figure 1(a) organizes these choices into three *search strategies* (direct revision, iterative refinement, and parallel sampling) and four *revision forms* (rules, examples, procedures, and rewrites). We call a search strategy together with the revision forms applied under it a *revision operator*, and the methods above each commit to a single one. Some prompt optimizers choose among editing operators adaptively (Ashizawa et al., 2025; Cui et al., 2025), though within one fixed search procedure.

No single revision operator is a global winner. We benchmark five of these methods under the same training budget. Figure 1(b) shows that the best baseline shifts with the benchmark and the target LLM: none of the existing methods ranks first in more than four of the eight benchmark-model combinations. The same instability appears within StraTune when its search strategy is fixed. LiveMath accuracy drops to 39.34 with direct revision (I1) alone and 29.38 with iterative refinement (I2) alone, against 65.88 when the strategy is chosen adaptively (Table 2). **The best revision operator also changes as optimization proceeds.** Each accept or reject reshapes what the skill still needs. An accepted candidate skill removes the failures it was written for, so the operator that just succeeded may not suit the failures that remain. A rejected one leaves the skill unchanged but signals that this kind of change is not what the remaining failures need. Trace2Skill observes the same under greedy patching, where a patch that fixes some tasks breaks others and a patch for an already covered failure only restates existing guidance (Ni et al., 2026).

We introduce **StraTune** (**stra**tegy-guided skill **tun**ing), which folds the choice of revision operator into the optimization (Figure 2). Each round, the frozen optimizer LLM reads the optimization state, which is the current execution feedback together with the recorded outcomes of earlier strategies and forms, then selects a search strategy and the revision forms used within it, and generates candidate skills (Figure 2(a)). Each strategy and form on this menu has a published precedent, shown in Figure 1(a) (Yuksekgonul et al., 2024; Agrawal et al., 2026; Yang et al., 2024; Zhao et al., 2024); our contribution is the menu itself and the per-round selection from it. Candidate skills from all three search strategies and four revision forms pass through one candidate evaluation, and every accept or reject is written back into the optimization state, so the next choice of strategy and form builds on what earlier choices achieved (Figure 2(b)). Promising candidate skills are saved along the way, and final skill selection chooses among them and the current skill (Figure 2(c)).

Across four challenging benchmarks and two model settings, StraTune ranks first in most (Table 1), exceeding the strongest baseline by at least 3 points in three settings and by 46% on LiveMath with Haiku. Ablations attribute the gain to adaptive selection, since random and fixed strategy choices lower scores on all four benchmarks.

We make three contributions. **(i) A design space of skill learning.** We organize existing methods by search strategy and revision form (Figure 1) and compare five of them under one training budget, finding that no single revision operator is best across benchmarks and target LLMs. **(ii) StraTune.** The optimizer LLM chooses the search strategy and revision form every round from the optimization

state, and one candidate evaluation assesses all candidate skills under common criteria and records their outcomes for later choices. **(iii) Evidence that adaptive selection drives the gain.** StraTune ranks first in most benchmark-model settings and outperforms random, scheduled, bandit, and fixed strategy choices. Its choice of operator differs across benchmarks and keeps shifting within a run (Section 4.3).

2 Problem Statement

Skill learning. A large language model performs a task under a *skill*, reusable text in its context. *Skill learning* improves this text from the model's own executions while its parameters stay frozen. In each *round*, the model executes a batch of training tasks under the current skill, a second language model (or the same) revises the skill from the scored executions, and the scores of the revised skill on further tasks decide whether it replaces the current one.

Formulation. Let the *target LLM* F_θ be the frozen model that performs the task, and the *optimizer LLM* G_ϕ the frozen model that rewrites and optimizes the skill. Given an input $x \in \mathcal{D}_{\text{train}}$ from a training set and a skill s, the target LLM produces a *trajectory* $\tau \sim F_\theta(\cdot \mid x, s)$, which is a response or a multi-step interaction. An *evaluator* r assigns it a score $r(x, \tau) \in [0, 1]$. The quality of a skill s is its expected score over the task distribution,

$$J(s) = \mathbb{E}_x\big[r(x, \tau)\big]. \tag{1}$$

Given $\mathcal{D}_{\text{train}}$, an *initial skill* s_0, and a *training budget* B (defined below), skill learning seeks a *final skill* $\hat{s}$ with J as high as possible, and $J(\hat{s})$ is estimated on a held-out set $\mathcal{D}_{\text{test}}$. Following the common practice of mini-batch training, here we split $\mathcal{D}_{\text{train}}$ into $N_{\mathcal{Q}}$ batches. At each round t, F_θ executes the *current skill* s_t on one batch $\mathcal{Q}_t$, and these executions with their scores form the *execution feedback* E_t. Based on s_t and E_t, G_ϕ writes *candidate skills* c. A *candidate evaluation* then executes each candidate c on training samples outside of $\mathcal{Q}_t$, compares its scores with those of s_t on the same samples, and decides whether the candidate replaces s_t as s_{t+1}. The main cost of skill learning is the executions of F_θ, and search strategies differ in how many executions a round uses, so stopping after a fixed number of rounds or epochs would give methods unequal compute. We therefore define the budget B as the total number of executions of F_θ on training samples, counting those for feedback and those for evaluating candidate skills. Test-set evaluation is not counted.

Our task. We study how candidate skills should be generated from (s_t, E_t) at each round. We call the procedure that produces them the *revision operator*. It combines a *search strategy*, which determines how candidates are generated, with *revision forms*, which determine what kind of text is written (three strategies I1–I3 and four forms F1–F4, Section 3.1). Existing methods fix one operator for the whole run, chosen before any feedback is seen. We instead choose the operator adaptively at every round. The operator o_t at round t is the search strategy used in that round together with the revision forms applied under it. It is chosen from the *optimization state* $\Omega_t = (s_t, E_t, H_t, p_t)$. Besides the current skill and its execution feedback, Ω_t holds the *evaluation history* H_t, which records the outcomes of earlier candidates grouped by strategy and by form, and p_t, which tracks a skill refinement in progress (see I2 in Section 3.1). The task is to choose o_t from Ω_t at every round, with ϕ fixed and no selection policy trained, so that $\hat{s}$ attains the highest J within B.

3 Method

Figure 2 gives the overview. At round t, G_ϕ chooses the revision operator from Ω_t and generates candidate skills (Section 3.2). A shared *candidate evaluation* compares them with those of s_t on training samples and records every outcome in H_{t+1}. After optimization, the *final skill selection* chooses $\hat{s}$ among the current skill and the *saved candidate skills* (Section 3.3).

3.1 Search Strategies and Revision Forms

Each strategy applies a form, so we describe the forms first. We consider four revision forms from existing work (Figure 2(a)): (1) *Conditional rules (F1)* give instructions for particular situations (Fu et al., 2024); (2) *Worked examples (F2)* pair a sample with its correct answer or a worked solution (i.e., few-shot examples (Brown et al., 2020)); (3) A *reasoning procedure (F3)* specifies reasoning steps or a workflow of task operations (Zhou et al., 2024); (4) A *full rewrite (F4)* revises the entire skill to incorporate new guidance and reorganize existing instructions. Details in Appendix A.

We define three alternative search strategies. (1) *Direct Revision (I1)* uses G_ϕ to propose c from the current skill s_t, choosing a revision form based on execution feedback E_t. (2) *Iterative Refinement (I2)* maintains an *intermediate skill* $\tilde{s}_t$, a separate working version initialized from s_t when

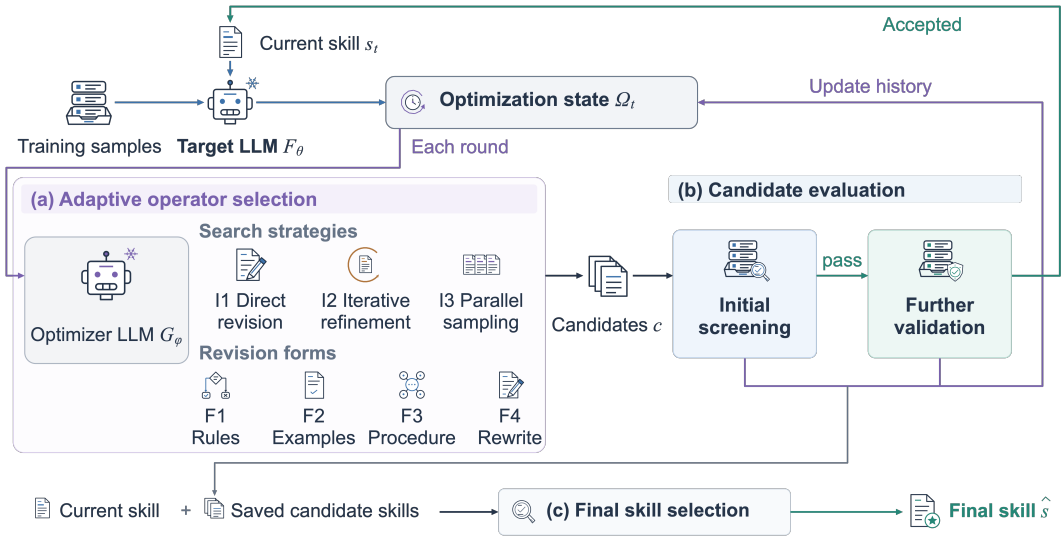


Figure 2: **Adaptive skill optimization with StraTune.** (a) From optimization state Ω_t, optimizer G_ϕ adaptively selects a revision operator o_t (a search strategy and the revision forms applied under it) and generates candidate skills, using execution feedback from target F_θ and task scores. (b) Candidate skills pass initial screening and then further validation; both stages record every outcome, including rejections, in the evaluation history H_t, and only candidates passing both replace the current skill. (c) After optimization, final skill selection compares the current skill with saved candidate skills to obtain $\hat{s}$.

refinement starts. At each step, G_ϕ proposes c by applying a selected revision form to $\tilde{s}_t$. An evaluated candidate skill may become the next intermediate skill even if it does not qualify to replace s_t. Refinement can therefore progress across rounds while the current skill remains unchanged. (3) *Parallel Sampling (I3)* uses G_ϕ to generate K candidate skills $c_1, \ldots, c_K$ from the same s_t under one selected revision form. It ranks these alternatives on a small training subset before submitting selected candidate skills for evaluation.

3.2 Adaptive Selection of Revision Operators

Given these strategies and forms, the question at each round is which revision operator to apply. StraTune lets G_ϕ choose the operator o_t from the optimization state Ω_t. The execution feedback E_t shows what the current skill still gets wrong, and the evaluation history H_t shows how earlier revisions with each strategy and form fared on similar failures. Both matter, because a strategy that failed on one kind of failure may suit another, and a strategy that succeeded may stop helping once the skill has changed.

Using the optimization state. The evaluation history H_t contains a *strategy history*, grouping candidate outcomes by I1–I3, and a *form history*, grouping them by explicitly selected revision forms. Both record, for each candidate skill, the attempt, the acceptance decision, the score change, and the *regressions*, losses on previously solved samples. They also record whether a candidate skill's gain on the small screening set held up on the larger validation set (the two stages of candidate evaluation, Section 3.3), which tells G_ϕ how far the gains of earlier revisions generalized.

For I2, p_t records whether refinement has started and how many steps have been completed. This lets G_ϕ consider continuing work on $\tilde{s}_t$, even when no candidate skill has yet replaced s_t.

Operator selection and candidate generation. G_ϕ is prompted to choose an available strategy suited to E_t and explain how the results in H_t support its choice. It also selects revision forms and generates candidate skills under that strategy. Forms can change across successive refinement steps in I2. The first round uses I1, since no evaluation history exists yet. Appendix A.3 gives the abbreviated selection prompt.

3.3 Candidate Evaluation and State Updates

Candidate evaluation decides whether a candidate c replaces s_t and, through its outcomes, supplies the evidence in H_{t+1} for later operator choices. Two failure modes shape its design. A positive mean gain can hide regressions on samples the current skill already solves, and a gain measured on

a few samples can vanish on more. Candidate evaluation therefore checks regressions explicitly and confirms gains first on a few samples and then on more.

Screening and validation. Each candidate skill is *submitted* to candidate evaluation as soon as it is generated, rather than at the end of the round. All submitted candidate skills are compared with s_t under the same procedure (Figure 2(b)). F_θ executes both skills on the same training subset $Q \subseteq \mathcal{D}_{\text{train}}$, and two measures decide the outcome. The first is the mean score gain of c over s_t, written $\widehat{\Delta}_Q(c, s_t)$. The second is the number of regressions, samples that s_t solves but c fails or scores substantially lower on. A candidate with more regressions than improvements is rejected.
We employ two evaluation stages, both of which apply this comparison and differ only in the sample set and the minimum gain: *1) Initial screening* uses a small set $Q_{\text{s}} \subseteq \mathcal{D}_{\text{train}}$ that includes previously solved samples and excludes samples from the current batch $\mathcal{Q}_t$ that are used to generate c, and requires $\widehat{\Delta}_{Q_{\text{s}}}(c, s_t) \geq \epsilon_{\text{s}}$; 2) *Further validation* is only applied to passing candidate skills on a larger *validation set* $Q_{\text{v}} \subseteq \mathcal{D}_{\text{train}}$, a random sample of training samples disjoint from Q_{s} and from $\mathcal{Q}_t$; no separate validation split is used. Validation requires $\widehat{\Delta}_{Q_{\text{v}}}(c, s_t) \geq \epsilon_{\text{v}}$ against the same s_t and tests whether the screening gain persists. Exact criteria are given in Appendix B.

Updating the optimization state. Only a candidate skill passing both stages replaces s_t as the current skill. Every outcome, including rejection, updates the evaluation history H_{t+1}, that is, the strategy and form histories. These histories record which strategies and forms were tried, their observed gains and regressions, and whether screening gains persisted. Together with the skill used for the next batch, its execution feedback, and refinement progress, H_{t+1} forms Ω_{t+1}. Rejected candidate skills therefore still supply evidence for later operator choices, even though they leave the current skill unchanged.

Final skill selection. Saved candidate skills include accepted versions and screened candidates whose gain was positive but below the acceptance threshold. After optimization, final skill selection compares saved candidate skills with the current skill on a common set of training samples (Figure 2(c)) and keeps a candidate only if it passes the gain and regression checks. The resulting skill is the final skill $\hat{s}$; Appendix B gives details.

4 Experiments

Our experiments answer four research questions (RQ). **RQ1** Does selecting the revision operator adaptively at each round outperform methods that commit to one? **RQ2** What does the per-round choice of strategy and form contribute, and what does it cost? **RQ3** Which revision operators does StraTune choose, and how do the skills evolve during optimization? **RQ4** Do the learned skills generalize to a stronger target LLM?

4.1 Experimental Setup

Tasks and model settings. We evaluate StraTune and five recent baselines on four real-world datasets: 1) DocVQA (Mathew et al., 2021) for document understanding, 2) LiveMathematician-Bench (LiveMath) (He et al., 2026) for mathematical reasoning, 3) Mind2Web (Deng et al., 2023) for offline element selection conditioned on annotated action histories, and 4) SpreadsheetBench (Ma et al., 2024) for spreadsheet manipulation. Following previous works, we report average normalized Levenshtein similarity (ANLS) for DocVQA (Biten et al., 2019; Mathew et al., 2021), answer accuracy for LiveMath (He et al., 2026), element accuracy averaged over tasks for Mind2Web (Deng et al., 2023; Wang et al., 2024b), and the fraction of tasks passing all workbook checks for SpreadsheetBench (Ma et al., 2024). The main model setting uses Claude Haiku 4.5 as the target LLM and Claude Sonnet 4.6 as the optimizer LLM. We also evaluate another setting with a Claude Opus 4.8 target LLM and a Claude Opus 5 optimizer. All model parameters remain frozen.

Baselines and training budget. We compare StraTune with five self-evolving skill baselines: TextGrad (Yuksekgonul et al., 2024), GEPA (Agrawal et al., 2026), SkillOpt (Yang et al., 2026), Trace2Skill (Ni et al., 2026), and Iter-CoT (Sun et al., 2024). Within each dataset and model setting, all methods share the same target and optimizer LLMs, initial skill, task interface, and test samples. Prior work shows that differences in the compute spent during optimization can affect comparisons of skill and memory modules (Hajimiri et al., 2026). Each run therefore allows $6|\mathcal{D}_{\text{train}}|$ target-LLM executions as the total training budget B (Section 2), including final skill selection and excluding optimizer calls and test evaluation.
We also present ablations of StraTune in Section 4.2. We change one element of StraTune per run under the same training budget. *I1 only*, *I2 only*, and *I3 only* fix the search strategy, *Rand.* samples

Table 1: Test scores (0–100) with Haiku and Opus as target LLMs under a training budget of $6|\mathcal{D}_{\text{train}}|$ target-LLM executions. Bold marks the highest score in each model setting.

Method	DocVQA	LiveMath	Mind2Web	SpreadsheetBench
Haiku setting	Target: Haiku 4.5	Optimizer: Sonnet 4.6		
Initial skill	47.88	29.38	46.09	40.00
GEPA	91.26	43.60	46.18	52.50
TextGrad	90.69	31.75	46.16	45.00
SkillOpt	91.87	45.02	40.55	50.71
Trace2Skill	90.62	33.18	46.18	46.79
Iter-CoT	73.42	36.97	**47.99**	48.57
StraTune	**92.12**	**65.88**	47.49	**56.43**
Opus setting	Target: Opus 4.8	Optimizer: Opus 5		
Initial skill	94.04	42.18	41.80	39.29
GEPA	95.26	78.20	50.50	**72.86**
TextGrad	94.37	52.61	50.32	71.43
SkillOpt	95.25	74.41	50.26	70.71
Trace2Skill	95.56	53.08	50.20	71.79
Iter-CoT	95.25	48.34	49.92	65.71
StraTune	**96.95**	**81.52**	**51.63**	**72.86**

a strategy at random each round, and *Rot.* cycles through I1, I2, and I3 in consecutive rounds. *Once* keeps the strategy that the optimizer LLM chooses at its first free decision, *Replay* follows round by round the strategy sequence recorded in the StraTune run of Table 1 in a new run with a different order of training batches, and *No hist.* withholds the strategy and form histories H_t from the strategy selection. *Bandit* replaces the optimizer LLM in the strategy choice by a multi-armed bandit rule over the outcomes recorded in H_t, either ϵ-greedy (Sutton & Barto, 1998), UCB1 (Auer et al., 2002), or Thompson sampling (Thompson, 1933). In the above ablation settings, the optimizer LLM still chooses the revision form adaptively, while *F4 only* keeps adaptive strategy selection but restricts every candidate skill to a full rewrite. We also conduct ablation studies on the candidate skill evaluation method: *No valid.* accepts candidate skills after initial screening alone, and *No final sel.* takes the current skill at the end of optimization as the final skill.

Implementation details. Training samples are split into $N_{\mathcal{Q}} = 16$ batches with the same mix of task types, and each round uses one batch. To pass initial screening, a candidate skill must raise the mean score over the current skill by at least ϵ_{s} (at least 0.001 on DocVQA and 0.0025 elsewhere) and pass the regression checks of Appendix B. Further validation applies its own gain and regression checks on a larger sample set, with $\epsilon_{\text{v}} = 0.01$. The calls to F_θ and G_ϕ use temperature zero to maintain reproducibility. Further details of datasets, baselines, and ablations are in Appendix C. We report variability across four runs with different random seeds of StraTune in Appendix D.3.

4.2 Main Results

Comparison with baselines (RQ1). StraTune is the best-performing method in Table 1. It achieves the highest score in six of the eight dataset–model settings, has the same highest score as GEPA on SpreadsheetBench with Opus, and is the second best on Mind2Web with Haiku. With Haiku/Sonnet, its largest gain is on LiveMath, where it scores 65.88 compared with 45.02 for SkillOpt. It also improves on the strongest baselines by 3.93 points on SpreadsheetBench and 0.25 points on DocVQA, and on Mind2Web it trails Iter-CoT by 0.50 points, with GEPA and Trace2Skill at 46.18. With Opus, StraTune exceeds the strongest baselines by 1.39, 3.32, and 1.13 points on DocVQA, LiveMath, and Mind2Web, respectively, and matches GEPA at 72.86 on SpreadsheetBench. The baselines are less consistent. The strongest baseline is GEPA, which achieves the best scores across all baselines in four settings, SkillOpt in two, and Iter-CoT and Trace2Skill in one each. Iter-CoT, the strongest on Mind2Web with Haiku, is the weakest on DocVQA with Haiku and on LiveMath, Mind2Web, and SpreadsheetBench with Opus. On DocVQA, the initial skill already scores 94.04 with Opus against 47.88 with Haiku, leaving less room for improvement. Taken together, each baseline commits to a single revision operator (Figure 1), and none of them performs consistently well across all tasks, whereas StraTune, which selects the revision operator adaptively, is the only method that ranks first or within 0.5 points of first in all eight settings.

Ablation of adaptive candidate generation (RQ2). Table 2 compares StraTune with the ablations of candidate generation defined in Section 4.1, all with the same candidate evaluation and training budget. **The best fixed search strategy depends on the task.** I1 alone is strongest on DocVQA,

Table 2: Test scores of ablations of candidate generation with Haiku/Sonnet (defined in Section 4.1).

Configuration	DocVQA	LiveMath	Mind2Web	SpreadsheetBench
StraTune	**92.12**	**65.88**	**47.49**	**56.43**
Fixed search strategy				
I1 only	91.92	39.34	46.09	54.64
I2 only	91.56	29.38	47.34	52.14
I3 only	90.73	63.03	45.99	55.00
Strategy chosen without the per-round decision				
Rand.	91.10	63.98	46.58	52.50
Rot.	91.74	65.40	46.06	52.14
Once	91.53	63.03	46.14	55.36
Replay	92.00	51.66	46.18	53.21
Strategy chosen by a bandit rule over the histories				
Bandit, ϵ-greedy	91.94	55.45	46.09	49.64
Bandit, UCB1	90.92	38.39	46.02	53.21
Bandit, Thompson	91.41	45.02	46.07	53.57
Strategy chosen without the histories				
No hist.	91.74	34.12	46.09	52.14
Fixed revision form				
F4 only	90.43	42.18	46.07	51.79

Table 3: Test scores of ablations of candidate evaluation with Haiku/Sonnet (defined in Section 4.1).

Configuration	DocVQA	LiveMath	Mind2Web	SpreadsheetBench
StraTune	**92.12**	**65.88**	**47.49**	**56.43**
No valid.	91.51	40.28	46.09	47.86
No final sel.	**92.12**	**65.88**	**47.49**	51.43

Table 4: Test scores with Haiku/Sonnet. The indented rows keep the candidate generation of GEPA or SkillOpt but use the candidate evaluation of StraTune.

Configuration	LiveMath	Spreadsheet Bench
GEPA	43.60	52.50
with StraTune evaluation	37.91	39.64
SkillOpt	45.02	50.71
with StraTune evaluation	52.13	53.93
StraTune	**65.88**	**56.43**



Figure 3: Training tokens of StraTune with Haiku/Sonnet on target-LLM executions (squares) and strategy selection (diamonds), and the latter as a percentage of the former.

I3 on LiveMath and SpreadsheetBench, and I2 on Mind2Web. StraTune exceeds each fixed strategy on all four datasets. **Removing the per-round decision lowers the score on every dataset.** *Rand.* is below StraTune by 1.02, 1.90, 0.91, and 3.93 points. *Rot.* stays within 0.5 points of StraTune on DocVQA and LiveMath and trails by 1.43 and 4.29 points on Mind2Web and SpreadsheetBench. *Once* trails by 0.59, 2.85, 1.35, and 1.07 points; on LiveMath, its chosen strategy was I3, so that run equals the I3-only score. *Replay* is below StraTune on every dataset, by 0.12, 14.22, 1.31, and 3.22 points. Its run draws different training batches and candidate skills, to which the copied sequence does not respond. **A bandit rule over the outcomes does not replace the optimizer LLM.** Choosing the strategy by ϵ-greedy, UCB1, or Thompson sampling from the same recorded outcomes scores below StraTune on all four datasets, by at most 1.5 points on DocVQA and Mind2Web and by 2.9 to 27.5 points on LiveMath and SpreadsheetBench. With at most two accepted candidate skills per run, their reward is almost always zero, so they either keep exploring or lock onto the first rewarded strategy, whereas the optimizer LLM also reads what the current skill gets wrong (Appendix D.2). **The histories inform the decision.** Withholding them from the strategy selection (*No hist.*) lowers the score by 0.38, 31.76, 1.40, and 4.29 points, most on LiveMath, where the run falls to 34.12. **The choice of revision form contributes on its own.** *F4 only* lowers the score on every dataset by 1.69, 23.70, 1.42, and 4.64 points, most on LiveMath, so full rewrites alone miss improvements that rules,

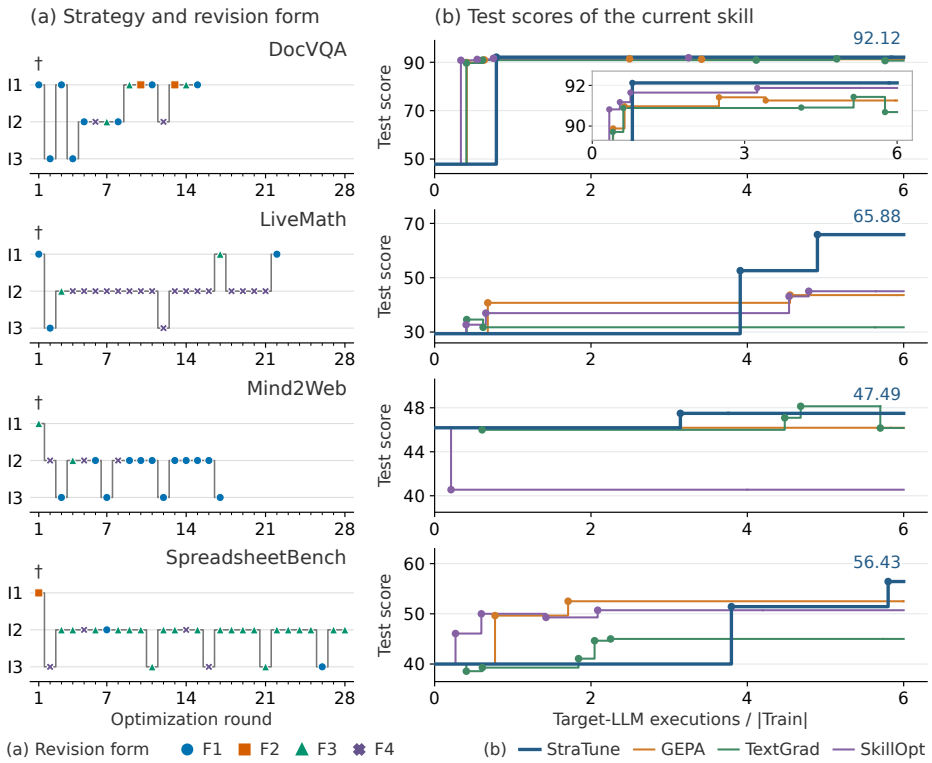


Figure 4: The StraTune runs of Table 1 with Haiku/Sonnet. **(a)** Strategy and revision form (marker) chosen in each round and † marks initialization. **(b)** Test score of the current skill against the number of F_θ executions divided by $|\mathcal{D}_{\text{train}}|$; filled circles mark changes of the current skill.

examples, and procedures express, and the choice of form should also stay with the optimizer LLM. Appendix D.1 fixes both at once.

Although some ablations come close on individual datasets, StraTune ranks first on all four benchmarks in Table 2. The strongest ablation differs across datasets, suggesting that adaptive selection helps maintain consistently strong performance across tasks.

Ablation of candidate evaluation and final skill selection (RQ2). Table 3 shows that both evaluation steps matter. Accepting candidate skills after initial screening alone (*No valid.*) lowers the score on all four datasets, most on LiveMath, from 65.88 to 40.28, and on SpreadsheetBench, from 56.43 to 47.86. Final skill selection confirms the current skill on three datasets and on SpreadsheetBench selects a candidate skill proposed in round 2 as the final skill, raising the score from 51.43 to 56.43 (*No final sel.*).

Baseline candidate generation with the candidate evaluation of StraTune (RQ2). Table 4 isolates the contribution of candidate evaluation. The two indented rows keep the candidate evaluation and final skill selection of StraTune and the same training budget, and differ only in who generates the candidate skills, GEPA or SkillOpt (Appendix C.2). In both cases the same evaluation yields a weaker skill than StraTune, so the gain cannot be attributed to evaluation alone. GEPA's reflective rewrites mostly fail the evaluation, so the current skill is rarely replaced and the scores fall below GEPA itself. This measures only GEPA's proposal step under our candidate evaluation; Table 1 compares GEPA's own search. SkillOpt's small edits pass more often, improving its LiveMath score from 45.02 to 52.13. Its scores on both datasets remain below StraTune and, on SpreadsheetBench, below *I1 only*. Candidate evaluation thus decides which candidate skills are kept, while the gain is set by how they are generated.

Token overhead of strategy selection (RQ2). Existing self-evolving skill methods spend their training cost on generating candidate skills with the optimizer LLM and evaluating them with the target LLM (Gao et al., 2025). StraTune adds one optimizer call per round for strategy selection and no target-LLM executions, and we measure this added cost in tokens on complete Haiku/Sonnet runs (Figure 3). The strategy-selection calls use 0.50, 0.76, 0.13, and 1.45 percent of the tokens spent on target-LLM executions on DocVQA, LiveMath, Mind2Web, and SpreadsheetBench. Strategy

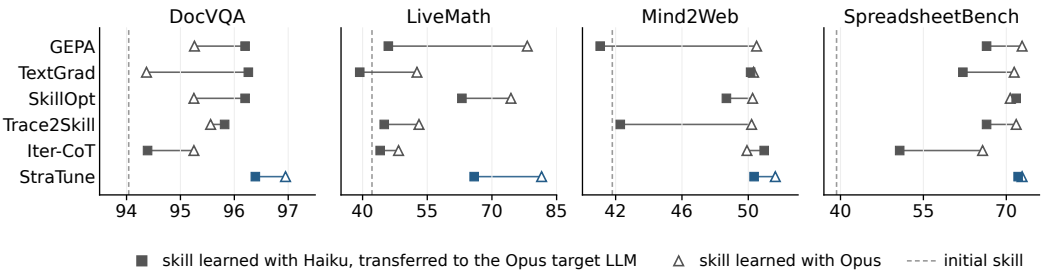


Figure 5: Test scores with Opus 4.8 as the target LLM of skills learned with Haiku (squares) and with Opus itself (triangles). The dashed line is the initial skill, and StraTune is highlighted by blue.

selection therefore adds a small token overhead relative to target-LLM executions. Appendix C.4 reports all training tokens used by the target LLM and optimizer LLM in these runs.

4.3 Discussion

Strategy and form choices (RQ3). Figure 4(a) shows the strategy and revision form StraTune chose in each round; the choice settles into a task-specific pattern yet keeps shifting within it. On LiveMath, 16 of the 22 rounds refine the intermediate skill through full rewrites (I2 with F4), whereas on SpreadsheetBench the same strategy edits reasoning procedures (I2 with F3) in 19 of 28 rounds. On DocVQA, direct revision (I1) is chosen in 8 of 15 rounds, proposing conditional rules (F1) in 4 of them and worked examples or reasoning procedures (F2 or F3) in the others, so the form varies even under one strategy. On LiveMath, Mind2Web, and SpreadsheetBench, parallel sampling (I3) appears in short bursts between longer sequences of I2 rounds, and the form it samples follows the task, conditional rules (F1) on Mind2Web and mostly full rewrites or reasoning procedures (F4 or F3) on SpreadsheetBench. **No two benchmarks follow the same sequence**.

Optimization trajectories (RQ3). Figure 4(b) plots the test score of the current skill after $k|\mathcal{D}_{\text{train}}|$ target-LLM executions. The baselines obtain most of their gain within the first $|\mathcal{D}_{\text{train}}|$ executions, and their later updates are small and sometimes harmful, as when SkillOpt's Mind2Web score drops to 40.55 at $0.21|\mathcal{D}_{\text{train}}|$ and never recovers. StraTune accepts one or two updates per benchmark and each raises the test score, although they arrive later, after $3|\mathcal{D}_{\text{train}}|$ or more on LiveMath, Mind2Web, and SpreadsheetBench. They arrive later because a candidate skill must pass both initial screening and further validation (Section 3.3).

Transfer to a stronger target LLM (RQ4). Final skills learned with Haiku also improve a stronger target LLM without further optimization. Figure 5 evaluates the Haiku final skills of every method in Table 1 with Opus 4.8 as the target. The StraTune skills raise the score of the initial skill on every dataset and recover 81, 60, 87, and 98 percent of the gain of the StraTune skills learned with Opus itself. The transfer is weakest on LiveMath, where the Haiku skill mainly teaches one recurring type of question and gains nothing on the rest, whereas the skill learned with Opus also improves them. On DocVQA, the Haiku skills of GEPA, TextGrad, SkillOpt, and StraTune score above every baseline skill learned with Opus itself. All four contain one rule, to answer with the value alone, for example *Dallas* rather than *Dallas, Texas*, because such format errors cause most failures of Haiku, whose initial skill scores 47.88. Opus starts from 94.04 and meets these errors too rarely in a training batch to learn the rule, yet they still make up two thirds of its remaining failures, so the rule learned with Haiku removes them. The changes these skills make therefore concern the task rather than the target LLM. StraTune can thus learn skills with an inexpensive target LLM and apply them to a stronger, more expensive one, reducing the cost of optimization.

5 Conclusion

We presented StraTune, which makes the choice of revision operator, a search strategy and the revision forms applied under it, part of skill optimization. The optimizer LLM selects the operator each round from the optimization state, and one candidate evaluation decides which candidates replace the current skill and records the outcome for later choices. StraTune ranks first in most of the eight benchmark-model settings, its ablations attribute the gain to choosing how to learn rather than to evaluation alone, and skills learned with Haiku also improve a stronger target LLM without further optimization.

Statement on the Use of Generative AI

Large language models are the object of study in this paper: Claude models serve as the target and optimizer LLMs of all methods, and every skill reported in the paper was produced by these models during optimization. In preparing the manuscript, the authors used Claude Fable 5 to edit text for grammar and check the consistency between manuscript and the code. All research questions, experimental designs, results, and claims were conceived, checked, and approved by the authors.

Ethics Statement

This work uses four publicly available benchmarks and involves no human subjects, personal data, or newly collected data. The learned skills are natural-language instructions produced by LLMs; they may contain errors or task-specific shortcuts, and we recommend reviewing them before use outside the evaluated tasks. SpreadsheetBench tasks execute generated programs, which we run in a sandboxed environment. We are not aware of conflicts of interest, and we adhere to the ICLR Code of Ethics.

Reproducibility Statement

The supplementary material contains the code of StraTune, the learned skills of Table 1, and a small sample of each benchmark (40 training and 20 test tasks) on which every stage of the code can be run. The full benchmark payloads exceed the supplementary material limit and will be provided upon publication. The method is specified in Section 3, with the search strategies, revision forms, and selection prompt in Appendix A.1–A.3 and the candidate evaluation rules in Appendix B. Appendix C lists the datasets and splits, model versions and decoding settings, the training budget, the baseline configurations, and the StraTune settings. All five baselines are run with their publicly released code, with the settings given in Appendix C.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, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026.

Rin Ashizawa, Yoichi Hirose, Nozomu Yoshinari, Kento Uchida, and Shinichi Shirakawa. Bandit-based prompt design strategy selection improves prompt optimizers. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20799–20817, 2025.

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

Ali Furkan Biten, Ruben Tito, Andres Mafla, Lluis Gomez, Marçal Rusinol, Ernest Valveny, CV Jawahar, and Dimosthenis Karatzas. Scene text visual question answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4291–4301, 2019.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Olivier Chapelle and Lihong Li. An empirical evaluation of Thompson sampling. *Advances in Neural Information Processing Systems*, 24, 2011.

Wendi Cui, Jiaxin Zhang, Zhuohang Li, Hao Sun, Damien Lopez, Kamalika Das, Bradley A Malin, and Sricharan Kumar. SEE: Strategic exploration and exploitation for cohesive in-context prompt optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29575–29627, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michele Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60(1): 25–64, 2010.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. *Advances in Neural Information Processing Systems*, 37: 119919–119948, 2024.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, 2024.

Sina Hajimiri, Masih Aminbeidokhti, Jose Dolz, Ismail Ben Ayed, Issam H Laradji, Spandana Gella, and Nicolas Gontier. Are online skill and memory modules always worth their tokens? A budget-constrained study of web agents. *arXiv preprint arXiv:2606.15017*, 2026.

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.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*, 35:22199–22213, 2022.

Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*, 2014.

Zeping Liu, Haiqi Xu, Ni Lao, Lauren Bennett, Liang Zhao, Mansour Raad, Hu Shao, Lynne Buie, Marshall Mueller, Cheng-Chia Huang, et al. Spatial CoT: A spatial concept transformation guided LLM reasoning framework for complex geospatial question answering. *International Journal of Geographical Information Science*, pp. 1–49, 2026.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. SpreadsheetBench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-Refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Gengchen Mai, Yiqun Xie, Xiaowei Jia, Ni Lao, Jinmeng Rao, Qing Zhu, Zeping Liu, Yao-Yi Chiang, and Junfeng Jiao. Towards the next generation of geospatial artificial intelligence. *International Journal of Applied Earth Observation and Geoinformation*, 136:104368, 2025.

Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. DocVQA: A dataset for VQA on document images. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 2199–2208. IEEE, 2021.

Johannes Moll, Jean-Philippe Corbeil, Jiazhen Pan, Martin Hadamitzky, Daniel Rueckert, Lisa Adams, and Keno Bressem. GRASP: Gated regression-aware skill proposer for self-improving LLM agents. *arXiv preprint arXiv:2605.29668*, 2026.

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.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, 2024.

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

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.

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

Jiashuo Sun, Yi Luo, Yeyun Gong, Chen Lin, Yelong Shen, Jian Guo, and Nan Duan. Enhancing chain-of-thoughts prompting with iterative bootstrapping in large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4074–4101, 2024.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic Cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

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

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. In *International Conference on Learning Representations*, 2024a.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

Weijia Xu, Andrzej Banburski-Fahey, and Nebojsa Jojic. Reprompting: Automated chain-of-thought prompt inference through Gibbs sampling. *arXiv preprint arXiv:2305.09993*, 2023.

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*, 2024.

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

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

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with Agent Skills. Anthropic Engineering Blog, https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, 2025. Published October 16, 2025.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026.

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

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

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V Le, Denny Zhou, Swaroop Mishra, Huaixiu S Zheng, et al. Self-Discover: Large language models self-compose reasoning structures. *Advances in Neural Information Processing Systems*, 37:126032–126058, 2024.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

A Search Strategies and Revision Forms

A.1 Search Strategies

Execution feedback can suggest several ways to improve a skill. The optimizer LLM G_ϕ may compare different changes, continue refining an intermediate skill, or generate several candidate skills within one revision form. We provide three search strategies to support these different needs. Each takes the current skill s_t and the execution feedback E_t from the optimization state Ω_t and submits candidate skills c to candidate evaluation (Appendix B).

Direct Revision (I1) prompts G_ϕ with s_t and E_t to generate a candidate skill c, choosing the revision form that fits the observed failures, for example a conditional rule (F1) or a worked example (F2). The candidate skill proceeds to candidate evaluation.

Iterative Refinement (I2) retains a separate working version, the intermediate skill $\tilde{s}_t$, for continued revision across rounds. It is initialized from s_t when refinement starts. Each refinement step applies one selected revision form to $\tilde{s}_t$: F1–F3 edit it locally, and F4 rewrites it. G_ϕ generates candidate skills from $\tilde{s}_t$, compares them on a common training subset, and submits higher-scoring candidates for candidate evaluation. An evaluated candidate skill may become the next intermediate skill $\tilde{s}_{t+1}$ even if it does not yet qualify to replace s_t. The progress variable p_t in Ω_t records whether refinement has started and how many steps it has completed, and selecting I2 again resumes it, allowing several rounds of revision before a candidate skill is accepted.

Parallel Sampling (I3) generates K candidate skills $c_1, \ldots, c_K$ from the same s_t and E_t under one selected revision form. Separate generation calls produce alternative wordings of the same kind of change. The K candidates are first scored on a small shared set of training samples. The highest-scoring candidate skill proceeds to candidate evaluation, and so does the runner-up if its score on these samples is no more than 0.02 below the current skill's, since the small shared set is too noisy to discard a near tie. This helps when the value of a modification depends on how it is expressed.

A.2 Revision Forms

Table 5 defines the four revision forms. Their use within each search strategy is described in Section 3.1.

Table 5: Revision forms used to generate candidate skills.

Form	Name	Content introduced or revised
F1	Conditional rules	Instructions for particular situations, such as converting quantities to a common unit before comparison.
F2	Worked examples	A sample paired with a correct answer or solution procedure that illustrates the intended behavior.
F3	Reasoning procedure	Steps for solving a task, including reasoning and answer checking or a workflow of task operations. Examples include comparing two derivations, or reading a table, filtering rows, and writing the result.
F4	Full rewrite	A rewrite of the entire skill to incorporate guidance, group related instructions, and resolve conflicts.

A.3 Adaptive Selection of Revision Operators

Figure 6 gives the abbreviated selection prompt used in Section 3.2. It supplies execution feedback E_t, descriptions of I1–I3, the strategy and form histories in H_t, and refinement progress p_t.

Under I1, G_ϕ selects a revision form while generating a candidate c from s_t. Under I2 and I3, it selects one form together with the search strategy. For I2, the selected form is applied to $\tilde{s}_t$ in the current refinement step: F1–F3 edit it locally, while F4 rewrites it. Successive steps can therefore use different forms. For I3, G_ϕ generates $c_1, \ldots, c_K$ from s_t using the selected form.

B Skill Evaluation Rules

This section states the evaluation criteria summarized in Section 3.3. Threshold schedules, early stopping, and maintenance of the pool of saved candidate skills follow the released implementation.

Paired comparison. Let c be a candidate skill and b the reference skill it is compared with; during candidate evaluation, $b = s_t$. The target LLM F_θ executes each sample x in an evaluation set Q under both skills. For an execution τ under skill s, the observed score is $\hat{r}(x; s) = r(x, \tau)$, with r

StraTune Revision Operator Selection Prompt

You are optimizing a reusable skill for a target model. Choose a search strategy and, where applicable, a revision form for the next iteration using the execution feedback and history below.

Execution feedback
[Summary of current failures, recent evaluation results, and the availability of supporting examples and successful trajectories.]

Strategy history
[Previous attempts, acceptance decisions, and observed gains or regressions, grouped by search strategy.]

Form history
[Previous attempts, acceptance decisions, and observed gains or regressions, grouped by explicitly selected revision forms.]

Refinement progress
[Whether I2 has started and how many refinement steps have been completed.]

Search strategies
I1 Direct Revision (one form for the candidate); I2 Iterative Refinement (one form per step); I3 Parallel Sampling (several candidates in one form).

Revision forms
F1 conditional rules; F2 worked examples; F3 reasoning procedure (reasoning steps or a workflow); F4 full rewrite.

Your task
Choose a search strategy that addresses the current failures, and the revision form where the strategy takes one, taking previous evaluation results into account. Briefly explain how the feedback and history support your choice.

Figure 6: An abbreviated prompt template for adaptive selection of revision operators in StraTune. The prompt uses *iteration* for what the text calls a round.

defined in Section 2. The mean gain is

$$\widehat{\Delta}_Q(c,b) = \frac{1}{|Q|}\sum_{x\in Q}\left[\hat{r}(x;c) - \hat{r}(x;b)\right]. \tag{2}$$

Regressions. Let $\mathcal{S}_b, \mathcal{S}_c \subseteq Q$ be the sets of samples solved with b and c according to the dataset's success criterion, let $d_x = \hat{r}(x;c) - \hat{r}(x;b)$, and let $\delta = 0.10$ mark a substantial change of a continuous score. We define

$$S = |\mathcal{S}_b|,\ D = \left|(\mathcal{S}_b \setminus \mathcal{S}_c) \cup \{x \in \mathcal{S}_b : d_x \le -\delta\}\right|,\ U = \left|(\mathcal{S}_c \setminus \mathcal{S}_b) \cup \{x \in Q : d_x \ge \delta\}\right|, \tag{3}$$

so D counts newly failed samples or large decreases among previously solved samples, and U counts newly solved samples or large increases; each sample is counted at most once in D or U. Two loss proportions are checked: D/S, the share of previously solved samples on which the candidate regressed, and $D/(D+U)$, the share of losses among all changes. Screening sets hold only 36 to 48 samples, so these proportions are noisy, and we test them through a confidence bound rather than the raw ratio. We write $\mathrm{LB}(k,n)$ for the one-sided 90% Wilson lower confidence bound (Wilson, 1927) on the true proportion when k of n samples are losses: with $\hat{p} = k/n$ and $z = 1.2816$, the 90% quantile of the standard normal distribution,

$$\mathrm{LB}(k,n) = \max\left(0,\ \frac{\hat{p} + \frac{z^2}{2n} - z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}\right), \qquad \mathrm{LB}(k,0) = 0. \tag{4}$$

$\mathrm{LB}(k,n)$ lies below $\hat{p}$ and approaches it as n grows, so a few losses on a small set do not by themselves fail a candidate. The two regression checks are $\mathrm{LB}(D,S) \le \frac{1}{2}$ and $\mathrm{LB}(D,D+U) \le \frac{1}{2}$: a check fails only when even the lower bound says that losses are a majority, and a check with no eligible samples passes.

Initial screening. The screening set $Q_s \subseteq \mathcal{D}_{\text{train}}$ combines samples previously solved with b and a stratified random sample of the remaining eligible samples, excluding the training batch and any recorded examples used to generate c. The candidate skill passes if $\widehat{\Delta}_{Q_s}(c,b) \ge \epsilon_s$, the number of samples on which c scores higher than b is at least the number on which it scores lower,

$\mathrm{LB}(D_\mathrm{s}, S_\mathrm{s}) \le \frac{1}{2}$, and $\mathrm{LB}(D_\mathrm{s}, D_\mathrm{s} + U_\mathrm{s}) \le \frac{1}{2}$, where the subscripts indicate the samples used. The minimum screening gain is $\epsilon_\mathrm{s} = \max(\epsilon_0, \tilde{m}/|Q_\mathrm{s}|)$, where ϵ_0 is a configured floor (Appendix C) and $\tilde{m}$ is the median absolute nonzero score difference between c and b on Q_s; for binary scores $\tilde{m} = 1$, so the candidate must gain at least one net sample.

Further validation. For each candidate skill, the validation set Q_v is a random sample of $\mathrm{round}(0.3\,|\mathcal{D}_\mathrm{train}|)$ training samples that excludes the candidate's screening samples and the batch used to generate it. Against the same reference skill b, the candidate skill must satisfy $\widehat{\Delta}_{Q_\mathrm{v}}(c, b) \ge \epsilon_\mathrm{v}$ and $\mathrm{LB}(D_\mathrm{v}, D_\mathrm{v} + U_\mathrm{v}) \le \frac{1}{2}$, where ϵ_v is the minimum validation gain. Table 7 lists the dataset-specific set sizes.

Saved candidate skills. Accepted candidates are saved, including earlier accepted versions replaced by later updates. Candidates that pass the regression checks with a positive gain below ϵ_s are also saved.

Final skill selection. After optimization, the current skill and the saved candidate skills are executed on a common set of training samples, which is split into a selection subset (60%) and a confirmation subset (40%). The two subsets are disjoint so that a candidate chosen for its gain on the selection subset is confirmed on samples that did not influence the choice; choosing and confirming on the same samples favours candidates whose measured gain is partly noise. The current skill is the selected skill at the start. Each saved candidate skill in turn is compared with the selected skill and replaces it only if it passes on both subsets. On the selection subset, its mean gain over the selected skill must be at least $\max(0.01, \tilde{m}/n)$, where n is the size of the subset and $\tilde{m}$ is defined as in initial screening, the number of samples on which it scores higher than the selected skill must be at least the number on which it scores lower, and $\mathrm{LB}(D, S) \le \frac{1}{2}$ and $\mathrm{LB}(D, D + U) \le \frac{1}{2}$ must hold. On the confirmation subset, the same conditions apply with a positive mean gain in place of the threshold. In this stage D counts only samples solved by the selected skill and failed by the candidate, without the δ term of Equation 3. The selected skill after the last comparison is the final skill $\hat{s}$.

C Experimental Details

C.1 Datasets and Execution Settings

Table 6 lists the fixed training and test sets used in Table 1. Any samples used for candidate generation, screening, validation, or final skill selection are drawn from the training set. Methods that require a fixed internal validation split construct it from this set.

Table 6: Dataset sizes and training budgets of target-LLM executions shared by all methods. One execution runs one skill on one training sample; a Mind2Web task counts once despite containing multiple action steps.

Dataset	Training	Test	Budget ($6\|\mathcal{D}_\mathrm{train}\|$)
DocVQA	500	1,000	3,000
LiveMath	397	211	2,382
Mind2Web	800	252	4,800
SpreadsheetBench	629	280	3,774

DocVQA. We use a fixed subset of 500 questions from the official training split and 1,000 from the labeled validation split, which serves as our test set. Sampling approximately preserves question-type frequencies and excludes overlap in document identity between the two sets. The target receives a page image and a question. We score the extracted answer with ANLS, taking the maximum over reference answers.

LiveMath. We use LiveMathematicianBench releases from November 2025 through April 2026 for training and May through June 2026 for testing, with no overlap in source papers. Each question has five answer choices. The correct choice and four distractors are deterministically shuffled before being labeled A–E. The target receives the question and choices without a proof sketch; the score is one if the predicted label is correct and zero otherwise.

Mind2Web. We use a fixed set of 800 training tasks and 252 tasks from the official cross-task test split. Evaluation predicts each action from the annotated prior actions and candidate skill page elements. Element accuracy is computed within each task and then averaged over tasks. This measures element selection under the given action history, rather than end-to-end success in a live browser.

SpreadsheetBench. Training uses 629 tasks from the 912-task release after excluding the test tasks and empty instructions. Testing uses the 280-task split of SpreadsheetBench-Verified released with SkillOpt (Yang et al., 2026). The target generates one Python program from the instruction and workbook preview; the program is executed and the output workbook checked against the references. A task passes only if all its workbook cases pass. Training tasks usually have three cases, whereas test tasks have one.

Models and decoding. The first model setting uses `claude-haiku-4-5-20251001-v1:0` for target executions and `claude-sonnet-4-6` for optimization. The second uses `claude-opus-4-8` and `claude-opus-5`, respectively. Target output limits are 6,000 tokens for LiveMath and 16,384 for the other tasks. A Mind2Web task can require several target calls, one per action step. The initial skill states the task and required response format. For example, the DocVQA initial skill reads: "You answer questions about document page images. You will be given a question and an image of a single document page. Answer the question using the content of the document image. Return the final answer inside `<answer>...</answer>` tags."

C.2 Baseline Settings

GEPA. We use the GEPA optimizer (Agrawal et al., 2026) with a single editable system-prompt component, Pareto candidate selection, and reflection minibatches of three samples. The merge operation is disabled. GEPA's system-aware merge composes a child by choosing, for each module, the prompt of one parent, so with a single component it can only reproduce a parent. GEPA also reports the merge-free optimizer as its main method and GEPA+Merge as a separate variant whose gains it attributes to systems with several modules. The fixed training set is divided 80/20 for generation and internal validation.

TextGrad. We implement the textual-gradient update procedure of TextGrad (Yuksekgonul et al., 2024) with minibatches of three samples. Each update is evaluated on the internal validation split and reverted if validation performance decreases. We use the same 80/20 partition rule as GEPA.

SkillOpt. We use the SkillOpt trainer (Yang et al., 2026) with task adapters for the shared execution environments. Training batches contain 40 samples and reflection minibatches 8, and the number of edits merged into the skill per step decays from four to two along a cosine schedule. Validation gating, slow updates, and optimizer-side meta guidance are enabled.

Trace2Skill. We use the combined variant of Trace2Skill (Ni et al., 2026), which extracts lessons from both successful and failed training trajectories and consolidates them into a skill. Three consolidation runs vary record order and grouping using seeds 41, 42, and 43. The final skill is selected by its score on the evolution samples, with a fallback to the initial skill if all candidate skills score lower. Task executions use the target LLM of the corresponding setting; analysis and consolidation use its optimizer LLM.

Iter-CoT. We follow the released pipeline of Iter-CoT (Sun et al., 2024), in which every construction call is made by the target LLM and no optimizer LLM is used. Each training sample is first answered with zero-shot chain of thought under the initial skill at temperature 0.7. Wrong answers are revised in the same conversation with the original revise prompt for up to six iterations, stopping early when an iteration corrects nothing, and the summary prompt then produces a complete reasoning process, kept only if its final answer is still correct. The final skill appends exemplars sampled once from this pool to the initial skill, eight on DocVQA and LiveMath and four on Mind2Web and SpreadsheetBench, without validation-based selection. On Mind2Web an exemplar is one action step. Revision calls count toward the training budget; summarization calls are logged but not counted.

Baseline candidate generation with the candidate evaluation of StraTune. The two indented rows of Table 4 replace the candidate generation of StraTune with that of a baseline and keep everything else. In each round, the generator receives the current skill and its execution feedback on the round's batch, exactly the executions that StraTune already pays for, and proposes candidate skills that enter the unchanged candidate evaluation, saved candidate skills, and final skill selection under the same $6|\mathcal{D}_{\text{train}}|$ budget. The GEPA generator is GEPA's reflective prompt mutation applied to a minibatch of three samples from the batch (Agrawal et al., 2026). GEPA's Pareto candidate pool and its minibatch-then-full evaluation are not used, because they perform the role that candidate evaluation performs here, and the parent is always the current skill of StraTune. The hybrid therefore isolates GEPA's proposal operator and does not evaluate GEPA's search as a whole. The SkillOpt generator is one SkillOpt step (Yang et al., 2026): its failure and success analysts propose

edits, which are merged, ranked under the cosine edit schedule of Appendix C.2, and applied to the current skill; its epoch-level slow update and its own selection gate are not used, since candidate evaluation takes the place of the latter. Both hybrids were run once on LiveMath and SpreadsheetBench.

C.3 StraTune Settings

Batches and set sizes. Training samples are split into 16 stratified batches. Table 7 lists the screening quotas and validation set sizes. I3 submits at most two of its three candidates.

Table 7: Screening quotas of previously solved and randomly sampled samples, and further-validation set sizes (30% of the training set), with Haiku/Sonnet.

	Screening		
Dataset	Solved	Random	Further validation
DocVQA	18	18	150
LiveMath	24	24	119
Mind2Web	18	18	240
SpreadsheetBench	18	18	189

The floor ϵ_0 of the screening gain is 0.001 on DocVQA and 0.0025 elsewhere and the validation gain ϵ_v is 0.01 (Appendix B); a sample counts as solved at a score of 0.9 on continuous-score task, and 1.0 on the binary-score tasks.

C.4 Token Accounting of StraTune

Table 8 breaks down the tokens that one complete StraTune training run consumes on each dataset with Haiku/Sonnet, counting the input and output tokens of every call and excluding test evaluation. Target tokens are spent by Haiku executing skills on training samples, for execution feedback, candidate evaluation, and final skill selection. Optimizer tokens are spent by Sonnet in three kinds of calls: strategy selection, the per-round choice of search strategy and revision form; candidate generation under I1, I2, and I3; and the preparation of execution feedback, which summarizes failures and successful trajectories for the generation calls.

Two observations follow. First, the optimizer LLM accounts for a small share of training tokens, between 1.1 and 22.7 percent of the target tokens; the largest share, on SpreadsheetBench, comes from feedback preparation. Second, strategy selection, the step that StraTune adds over methods with a fixed revision operator, uses between 6 and 11 percent of the optimizer tokens and at most 1.45 percent of the target tokens, the shares reported in Figure 3.

Table 8: Tokens consumed by one StraTune training run per dataset with Haiku/Sonnet. Target and optimizer tokens are in millions, strategy-selection tokens in thousands. Ratio is optimizer tokens as a percentage of target tokens.

	All optimizer calls			Strategy selection			
Dataset	Target	Optimizer	Ratio (%)	Calls	Tokens	Of optimizer (%)	Of target (%)
DocVQA	9.75	0.51	5.3	14	49	9.5	0.50
LiveMath	6.47	0.51	7.9	10	49	9.6	0.76
Mind2Web	45.00	0.50	1.1	16	57	11.3	0.13
SpreadsheetBench	7.20	1.63	22.7	25	104	6.4	1.45

D Additional Results

D.1 Fixed Revision Operators on LiveMath

Motivation. The ablations in Table 2 fix one of the two choices, the search strategy or the revision form, and leave the other to the optimizer LLM. They therefore do not compare adaptive selection with a complete revision operator held fixed. This experiment closes that gap by fixing both choices at once.

Setup. We run StraTune on LiveMath with Haiku/Sonnet twelve times, each time allowing exactly one revision operator: one search strategy (I1, I2, or I3) paired with one revision form (F1, F2, F3, or F4). Candidate evaluation, saved candidate skills, final skill selection, and the training budget of $6|\mathcal{D}_{\text{train}}|$ are unchanged, so these runs differ from the adaptive run only in that the operator is fixed. Worked examples (F2) need solved training samples as material, so rounds without such samples are skipped for the F2 operators. Each operator is run once.

Table 9: LiveMath test scores of StraTune with one fixed revision operator (rows: search strategy, columns: revision form), compared with the initial skill and with adaptive selection of strategy and form (Table 1); Haiku/Sonnet, $6|\mathcal{D}_{\text{train}}|$ budget.

	F1 rules	F2 examples	F3 procedure	F4 rewrite
I1 direct revision	46.92	45.50	43.13	34.12
I2 iterative refinement	28.44	29.38	29.38	29.38
I3 parallel sampling	37.91	36.49	63.51	58.29
Initial skill		29.38		
StraTune, adaptive strategy and form		65.88		

Results. Table 9 reports the test scores. First, the best fixed operator is parallel sampling of reasoning procedures (I3 with F3), which scores 63.51, close to the 65.88 of adaptive selection. Second, every other operator falls well short: I3 with F4 by 7.6 points and the remaining ten by 19 to 37 points. The four I2 operators never change the initial skill, because iterative refinement restricted to one form produced no candidate skill that passed candidate evaluation. Third, adaptive selection reaches the level of the best fixed operator without knowing which operator that is. Identifying it here required all twelve runs, and Table 2 shows that the best fixed strategy differs across datasets, so the operator cannot be chosen in advance. The adaptive run did not itself use I3 with F3; it obtained its score through iterative refinement with full rewrites (I2 with F4), so more than one operator leads to a strong skill on LiveMath.

D.2 Bandit Strategy Selectors on LiveMath and SpreadsheetBench

Motivation. The ablations *Rand.*, *Rot.*, *Once*, and *Replay* in Table 2 remove the per-round decision, but none of them selects the strategy from the recorded outcomes by a rule. The *Bandit* rows of Table 2 answer whether such a rule does as well as the optimizer LLM, which reads the failure types in the execution feedback in addition to the outcomes; this section gives the rules and the runs.

Setup. We replace the strategy choice of StraTune by three multi-armed bandit rules (Kuleshov & Precup, 2014), as in adaptive operator selection for evolutionary algorithms (Fialho et al., 2010), that see only the outcomes recorded in H_t: ϵ-greedy on the mean reward of each strategy with $\epsilon = 0.1$ (Sutton & Barto, 1998), UCB1 (Auer et al., 2002), and Thompson sampling with a Beta prior (Thompson, 1933; Chapelle & Li, 2011). The reward of a round is 1 if a candidate submitted by the chosen strategy replaced the current skill, 0.5 if it was saved for final skill selection, and 0 otherwise; untried strategies are chosen first. Everything else is unchanged: the same three strategies and four forms, the optimizer LLM still chooses the revision form, and candidate evaluation, saved candidate skills, final skill selection, and the $6|\mathcal{D}_{\text{train}}|$ budget are those of StraTune. We run each selector once on each dataset with Haiku/Sonnet, drawing the same training samples as the *Replay* runs of Table 2 and the *Rot.* runs on DocVQA, LiveMath, and Mind2Web, so these rows are directly comparable; the SpreadsheetBench *Rot.* run with the same samples scores 45.71 (Table 2 reports a second run, 52.14).

Table 10: Test scores with the strategy chosen by the optimizer LLM (StraTune), by a fixed or replayed schedule, and by three bandit rules over the recorded outcomes; Haiku/Sonnet, $6|\mathcal{D}_{\text{train}}|$ budget. The schedule and bandit rows use the same training samples.

Strategy selection	DocVQA	LiveMath	Mind2Web	SpreadsheetBench
StraTune, optimizer LLM (Table 1)	**92.12**	**65.88**	**47.49**	**56.43**
Rot., fixed schedule	91.74	65.40	46.06	45.71
Replay, copied sequence	92.00	51.66	46.18	53.21
ϵ-greedy	91.94	55.45	46.09	49.64
UCB1	90.92	38.39	46.02	53.21
Thompson sampling	91.41	45.02	46.07	53.57
Initial skill	47.88	29.38	46.09	40.00

Results. Table 10 reports the test scores. All three bandit rules score below StraTune on all four datasets, by 0.2 to 1.2 points on DocVQA, 10.4 to 27.5 on LiveMath, 1.4 to 1.5 on Mind2Web, and 2.9 to 6.8 on SpreadsheetBench, and no bandit exceeds the best fixed or replayed schedule on any dataset. DocVQA does not separate the selectors: every bandit run accepts the same round-1 candidate as the other methods, a conditional rule from direct revision, and the runs end within 1.2 points of one another. On Mind2Web no bandit run accepts a candidate that survives to the end, and final skill selection returns the initial skill in all three runs, whereas StraTune accepts one

update. The reason is visible in the runs. A run accepts zero to two candidate skills, so the reward a bandit receives is almost always zero and the rules fall back to their default behaviour. UCB1 keeps exploring and declares the three strategies almost uniformly (6, 6, and 5 rounds on LiveMath), and scores lowest on LiveMath. ϵ-greedy commits to the first strategy that is rewarded, direct revision, which it declares in 18 of 22 rounds on SpreadsheetBench and in 27 of 30 rounds on Mind2Web, and scores lowest on SpreadsheetBench. Thompson sampling lies between the two. The optimizer LLM receives the same outcomes but also reads what the current skill gets wrong, and Table 2 shows that withholding the outcomes from it (*No hist.*) also lowers the score. Both sources of information therefore matter, and a rule that sees only the outcomes does not replace the optimizer LLM at this budget. These are single runs, and LiveMath scores vary between runs of every configuration, so the LiveMath margins are indicative; the margins on the other three datasets are smaller but consistent across the three rules.

D.3 Variability Across Optimization Runs

We assess the variability of StraTune using four independent runs with different seeds for each dataset–model setting. Table 11 reports the mean and standard deviation of the test scores. The standard deviations range from 0.40 to 1.16 points on the 0–100 scale. The main comparison in Table 1 reports single-run results.

Table 11: Mean $\pm$ standard deviation of StraTune test scores (0–100) over four optimization runs per dataset–model setting.

Setting	DocVQA	LiveMath	Mind2Web	SpreadsheetBench
Haiku	92.52 ± 0.80	65.40 ± 1.16	47.52 ± 0.61	56.16 ± 0.90
Opus	96.88 ± 0.40	81.16 ± 0.80	51.48 ± 0.56	72.41 ± 0.70