THINK LESS, EVOLVE BETTER: UNLOCKING GATE-FREE SKILL SELF-EVOLUTION IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Skill self-evolution enables language model agents to refine reusable instructions from execution experience without updating model parameters. Existing methods often rely on costly validation gates to filter harmful revisions, yet removing these gates allows harmful updates to accumulate. We observe that accuracy-degrading revisions often lengthen reasoning, while fixed guidance against redundant reasoning mitigates degradation during ungated evolution. These observations suggest that reasoning efficiency can identify harmful guidance from existing execution trajectories. Motivated by this insight, we propose **FreeEvo**, a framework that eliminates validation gates by turning reasoning efficiency feedback into instruction selection. FreeEvo decomposes each skill into task-specific accuracy atoms and a fixed efficiency atom that discourages redundant reasoning. The optimizer uses this guidance to attribute unnecessary reasoning, useful procedures, and task errors to individual accuracy atoms. Evidence accumulated across distinct examples then determines atom activation and permanent retirement without additional candidate validation rollouts. Across three models and seven benchmarks spanning four task domains, FreeEvo matches or outperforms existing SOTA methods while reducing evolution tokens by up to 93.6%. At test time, the evolved agent improves task performance by up to 23.1 percentage points while using 6.4%–77.9% fewer output tokens. Our code will be released upon publication.

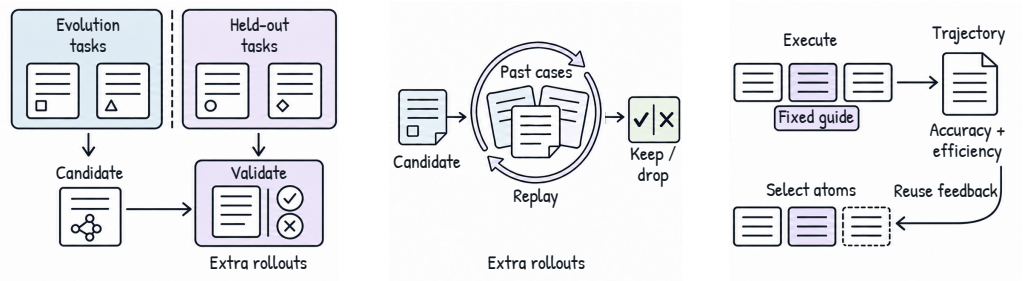
1 INTRODUCTION

Advances in agentic reinforcement learning (Xia et al., 2026; Wei et al., 2025) and execution harnesses (Yang et al., 2024) have expanded the capabilities of large language model (LLM) agents on complex tasks (Li et al., 2026b). Agent skills provide a lightweight way to adapt these capabilities to specific domains by encoding reusable procedures as human-readable instructions and executable workflows, without updating model parameters (Yang et al., 2026). However, skills are often manually written or distilled once from execution trajectories. Once fixed, they cannot incorporate lessons from new task variants or failure cases. As a result, corrections made during one interaction remain local to that session, allowing the same mistakes to recur in future tasks (Ma et al., 2026b).

Recent work addresses this limitation through skill self-evolution, which converts execution feedback into persistent skill updates. Although methods differ in their feedback, revision, and selection procedures (Liu et al., 2026), most introduce a *validation gate* that evaluates candidate skills before retention and accepts only those meeting predefined criteria. Based on the source of validation cases, these gates generally take two forms. A *held-out gate* (Yang et al., 2026; Li et al., 2026a) retains candidates that improve performance on held-out tasks, sometimes under additional regression constraints (Fig. 1(a)). A *replay gate* (Moll et al., 2026; Wang et al., 2026b) instead re-executes previously encountered cases to verify that candidates preserve existing capabilities (Fig. 1(b)). Yet these gates account for over 60% of the evolution cost on average (Fig. 2(a)), raising a fundamental question: *why is the gate so prevalent in skill self-evolution despite its substantial cost?*

Motivation. To understand this reliance, we remove the gate from existing state-of-the-art (SOTA) methods (Yang et al., 2026) while retaining their skill update procedure, so that every proposed update is accepted without extra validation. With the gate, accuracy improves at a few steps and the gains persist between accepted updates. Without it, accuracy fluctuates and eventually falls below

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.



(a) Held-out gate **(b) Replay gate** **(c) Gate-free (ours)**

Figure 1: **Skill self-evolution w/ and w/o validation gates.** (a) Held-out gates evaluate candidate skills on separate tasks. (b) Replay gates re-execute past cases to detect regressions. Both require extra rollouts. (c) FreeEvo reuses accuracy and reasoning-efficiency feedback from evolution trajectories to select skill atoms without extra rollouts, while keeping a fixed efficiency atom active.

its initial level on both tasks (Fig. 2(b)). These contrasting trajectories indicate that the gate prevents harmful revisions from accumulating, revealing the instability of the underlying update process. Avoiding its substantial cost therefore requires a signal that can identify harmful guidance directly from the execution feedback already collected during evolution.

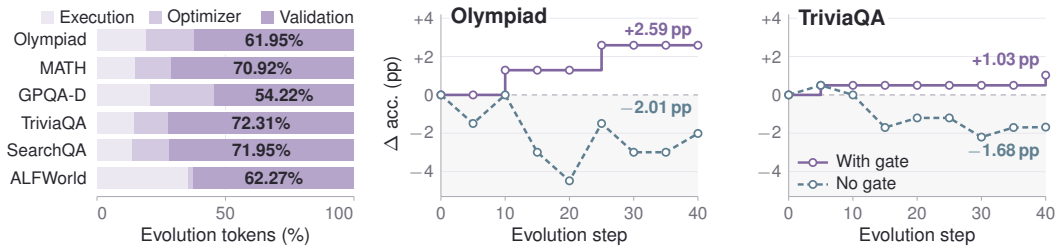
To seek such a signal, we consider a basic property of useful skills: they should provide reusable procedures that reduce redundant exploration and guide the agent toward effective reasoning trajectories (Zhao et al., 2026). Conversely, harmful guidance may induce unnecessary reasoning, making reasoning efficiency a natural signal for distinguishing beneficial from harmful updates. To test this hypothesis, we examine accuracy-degrading revisions rejected by the validation gate and find that most also increase output length (Fig. 3(a)). We further add simple fixed guidance discouraging redundant reasoning during ungated evolution. Although it does not directly optimize accuracy, it shortens outputs and substantially reduces the final accuracy decline (Fig. 3(b)). Together, these observations suggest that *harmful updates often manifest as less efficient reasoning, making reasoning efficiency an actionable signal for constraining skill evolution without a validation gate*.

Our solution. In this work, we propose **FreeEvo**, a gate-free skill self-evolution framework that translates reasoning-efficiency feedback into fine-grained skill updates. To make harmful guidance identifiable and removable, we first decompose a skill into modular *skill atoms*. Accuracy atoms encode task-specific procedures and their conditions of use, while a fixed efficiency atom guides the agent to avoid redundant reasoning. This formulation makes individual instructions the units of evolution and provides a persistent reference for assessing their effects on reasoning efficiency.

To determine which atoms should remain, we then introduce an evidence-driven update mechanism. An optimizer model[1] analyzes execution trajectories collected during evolution, using output-length changes to locate potential inefficiency and the efficiency atom as a reference for attributing redundant reasoning to specific accuracy atoms. Together with evidence of useful procedures and task errors, these judgments accumulate across examples to determine which atoms are activated or retired. The selected accuracy atoms are then combined with the fixed efficiency atom to form the skill for the next execution batch. This feedback loop extends efficiency guidance from task execution to instruction selection, constraining harmful updates without additional candidate-validation rollouts.

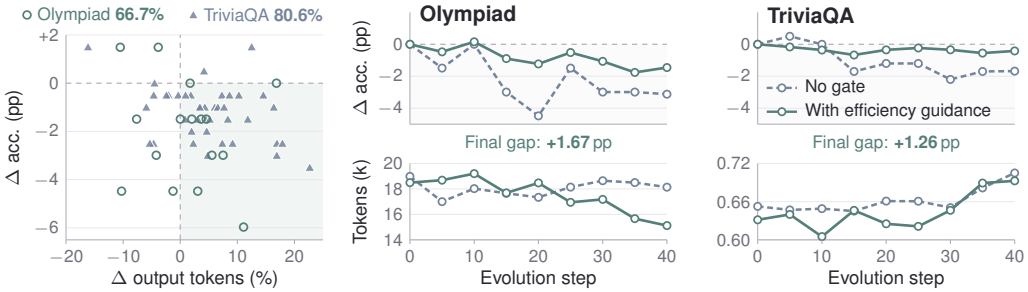
Experiments across three models and seven benchmarks covering mathematical reasoning, scientific question answering, reading comprehension, and embodied reasoning show that FreeEvo makes skill evolution substantially cheaper while producing more capable and efficient agents. Specifically, FreeEvo matches or outperforms existing SOTA methods while reducing evolution tokens by up to 93.6%. At test time, the evolved agent improves task performance by at most 23.1 percentage points while using 6.4%–77.9% fewer output tokens. To summarize, our contributions are as follows:

[1]The *execution model* is the LLM that performs tasks using the current skill. The *optimizer model* is the LLM that analyzes execution feedback to revise the skill instructions.



(a) Evolution token cost (b) Gate ablation

Figure 2: **Cost decomposition and gate ablation.** (a) Candidate validation consumes most of the evolution budget, averaging 65.6% across six benchmarks. (b) Gated evolution preserves occasional accuracy gains, while removing the gate leads to net degradation on both tasks.



(a) Update-level effects (b) Fixed efficiency guidance

Figure 3: **Analysis of candidate updates and fixed efficiency guidance.** (a) Most accuracy-degrading updates also increase output length, linking harmful revisions to inefficient execution. (b) Fixed efficiency guidance shortens outputs and reduces accuracy degradation.

- We reveal that reasoning efficiency reflects skill-update quality, as harmful revisions often lengthen execution and discouraging redundant reasoning mitigates degradation during ungated evolution. This makes efficiency an actionable signal for skill selection.
- We propose FreeEvo, a gate-free self-evolution framework that turns reasoning-efficiency feedback into instruction-level skill selection. It reuses execution evidence to retain useful accuracy atoms and retire harmful ones without additional candidate-validation rollouts.
- Experiments across three models and seven benchmarks spanning four task domains demonstrate substantial reductions in evolution cost together with improved task performance and more efficient test-time execution.

2 Background and Motivation

2.1 Preliminaries

In the following, we first describe how skills guide task execution and how execution feedback drives skill updates. We then formalize the role of the validation gate in this process.

Agent skills and self-evolution. A skill s encodes reusable task instructions supplied in an agent's prompt. Given a task x, the execution model π_θ generates a trajectory $\tau_x^s \sim \pi_\theta(\cdot \mid x, s)$ recording its reasoning, response, and any actions and observations. Skill self-evolution revises these instructions using execution feedback while keeping θ fixed. Starting from an initial skill s_0, step t executes a batch $\mathcal{B}_t$ from the evolution dataset $\mathcal{D}_{\text{evo}}$ with s_t, producing trajectories $\tau_{x,t}$. Let $r_{x,t} = R(x, \tau_{x,t}) \in \{0, 1\}$ and $\ell_{x,t} = L(\tau_{x,t})$ denote task success and output token count. An optimizer model $\mathcal{U}$ analyzes this feedback to identify useful procedures and errors, then proposes a candidate *revision* by adding, removing, or refining instructions:

$$\mathcal{F}_t = \{(x, \tau_{x,t}, r_{x,t}, \ell_{x,t}) : x \in \mathcal{B}_t\}, \qquad \widetilde{s}_{t+1} = \mathcal{U}(s_t, \mathcal{F}_t). \tag{1}$$

Here $\mathcal{F}_t$ contains the current batch's execution feedback, and $\widetilde{s}_{t+1}$ is the revised skill proposed for subsequent use. An update rule determines whether this candidate replaces s_t.

Validation gates and evolution cost. A *validation gate* determines whether the proposed revision updates the skill. Before adoption, the execution model evaluates the candidate on tasks $\mathcal{V}_t$ through additional rollouts. For example, a gate requiring higher accuracy applies

$$s_{t+1} = \begin{cases} \widetilde{s}_{t+1}, & \widehat{R}_{\mathcal{V}_t}(\widetilde{s}_{t+1}) > \widehat{R}_{\mathcal{V}_t}(s_t), \\ s_t, & \text{otherwise}, \end{cases} \tag{2}$$

where $\widehat{R}_{\mathcal{V}}(s) = |\mathcal{V}|^{-1} \sum_{x \in \mathcal{V}} R(x, \tau_x^s)$ is the observed accuracy. A *held-out gate* (Yang et al., 2026) uses a separate selection split, while a *replay gate* (Moll et al., 2026) re-executes previously encountered tasks. The accepted candidate, or unchanged skill after rejection, guides the next batch. This execution, revision, and validation cycle constitutes one *evolution step*. One *epoch* covers $\mathcal{D}_{\text{evo}}$ once. After evolution, the selected skill is frozen for testing. Over T steps, the total token cost is

$$C_{\text{evo}} = \sum_{t=0}^{T-1} \left(C_t^{\text{exec}} + C_t^{\text{update}} + C_t^{\text{gate}} \right), \tag{3}$$

where the terms count token usage for task execution, optimizer revision, and candidate validation.

2.2 Key Observations

As discussed above, validation gates are widely used in skill self-evolution. To quantify their cost, following Yang et al. (2026), we adopt a 2:1:7 training/validation/test split across six benchmarks. We run four epochs with 40 evolution steps and separately track tokens for execution, update generation, and validation. Validation accounts for 54.2%–72.3% of the total cost, averaging 65.6% (Fig. 2(a)). Given this substantial overhead, we next examine why validation gates remain widely used by assessing their role in filtering harmful updates before retention.

Validation gates stabilize an unreliable update process. We first compare SkillOpt with and without its gate on Olympiad (He et al., 2024) and TriviaQA (Joshi et al., 2017). Both variants start from the best skill selected on the original validation set and share identical training examples, batch order, models, and update settings. To avoid reusing data from prior selection, we sample a fresh validation set of the same size from each test split for gated acceptance and trajectory evaluation. Both variants evolve for 40 steps, and Fig. 2(b) reports accuracy changes from the initial skill every five steps. Without the gate, accuracy fluctuates and ultimately drops by 2.01 and 1.68 percentage points, showing that harmful revisions can accumulate under indiscriminate acceptance. Each gated run instead accepts only two updates, whose gains persist thereafter. These results show that validation is critical for preventing harmful updates, but its high cost motivates an actionable signal that identifies harmful guidance from execution feedback already available during evolution.

Reasoning efficiency offers a signal for constraining harmful updates. To seek such a signal, we consider a basic property of useful skills. By providing reusable procedures, they guide agents toward effective reasoning paths (Zhao et al., 2026) and reduce redundant exploration and reflection associated with overthinking (Chen et al., 2024b). Harmful guidance may instead induce unnecessary reasoning. We therefore examine whether reasoning efficiency can reveal harmful updates. As shown in Fig. 3(a), we compare SkillOpt candidate updates (Yang et al., 2026) with their incumbent skills in validation accuracy and output length. Among accuracy-degrading updates rejected by the gate, 66.7% and 80.6% also increase output length, indicating that harmful updates often reduce reasoning efficiency. Unlike validation accuracy, efficiency can be derived from existing evolution trajectories without additional candidate executions. This motivates our hypothesis that promoting efficient reasoning can mitigate harmful updates during ungated evolution.

To test this hypothesis, we use GPT-5.5 to generate task-specific efficiency guidance for Olympiad and TriviaQA (Appendix C.5) that discourages further reasoning once the answer is sufficiently supported. We keep this guidance fixed throughout 40 steps of ungated evolution while the remaining instructions evolve. Compared with standard ungated evolution, efficiency guidance reduces

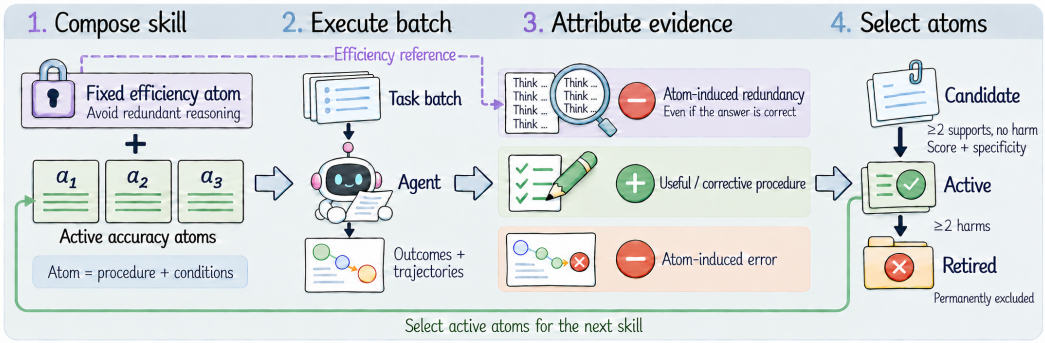


Figure 4: **Overview of FreeEvo.** (1) Active accuracy atoms and a fixed efficiency atom form the skill. (2) The agent executes a task batch, producing outcomes and trajectories. (3) The optimizer attributes useful procedures, errors, and redundant reasoning to accuracy atoms, with the efficiency atom guiding redundancy assessment. (4) Accumulated evidence determines atom activation and retirement, and selected active atoms form the next skill without additional validation rollouts.

the final accuracy decline by 1.67 and 1.26 points, while also shortening outputs (Fig. 3(b)). Together, these results show that reasoning efficiency can expose harmful updates and mitigate their effects. However, fixed guidance cannot prevent degradation from other evolving instructions. This motivates extending efficiency from execution guidance to instruction selection, using trajectory evidence to identify and remove harmful instructions without additional validation rollouts.

3 Method

3.1 Overview

Building on Sec. 2.2, FreeEvo uses reasoning-efficiency feedback to identify and remove harmful skill instructions without a validation gate. We first define a modular skill representation that separates task-specific procedures from efficiency guidance (Sec. 3.2). We then describe how these atoms are generated and how execution evidence determines which accuracy atoms are activated or retired (Sec. 3.3). Finally, we combine the selected atoms into a skill for subsequent execution (Sec. 3.4). Fig. 4 illustrates this loop, and Appendix A gives the detailed rules and full algorithm.

3.2 Modular Skill Representation

FreeEvo represents a skill as a collection of *skill atoms*. Each atom $a = (u, \mathcal{C})$ contains an instruction u specifying what to do and conditions $\mathcal{C}$ describing when to use it. A skill can be split into atoms according to its distinct procedures and guidance. Conversely, concatenating the instructions and conditions of selected atoms forms a skill. This allows individual instructions to be retained or removed without rewriting the remaining content.

We distinguish two types of atoms by their content. An *accuracy atom* a_i encodes a task-specific procedure intended to help solve the task correctly, together with its conditions of use. An *efficiency atom* a^{eff} encodes guidance for avoiding redundant reasoning within the benchmark's domain. Both contain an instruction and conditions of use. Appendix A.6 illustrates the two types with a mathematical reasoning example.

3.3 Skill Atom Generation and Evolution

FreeEvo generates and selects these atoms using the batch execution feedback $\mathcal{F}_t$ defined in Sec. 2.1.

Generating the efficiency atom. The first batch is executed using only the standard task prompt, without any evolved skill. Based on its feedback, the optimizer generates a single domain-specific efficiency atom. Zhao et al. (2026) show that reasoning skills generated once from execution trajectories can improve accuracy and reduce reasoning tokens when reused on new problems. Building

on this finding, we generate the efficiency atom only once and keep its instruction and conditions fixed from the next batch onward. The atom targets recurring reasoning inefficiencies within the domain, such as repeated verification after the relevant uncertainty has been resolved. The generation prompt is provided in Appendix D.1.

Generating accuracy atoms. Accuracy atoms are generated only during the first epoch, from the first to the last batch. After each batch, the optimizer examines current and stored trajectories to identify reusable procedures supported by recurring patterns across distinct examples. Each proposal is compared with existing atoms to ensure a distinct role and clear conditions of use. New atoms are initialized as *candidate* and become active only after accumulating sufficient evidence under the rule below. Once evidence is assigned, their instructions and conditions are fixed to ensure consistent evaluation thereafter. Subsequent epochs only continue evaluating these atoms. Appendix A.2 provides the discovery and refinement details.

Attributing evidence and selecting atoms. After each batch, the optimizer assesses each accuracy atom against the execution trajectories. For an example x, task-outcome evidence $v_{i,t}^{\mathrm{acc}}(x)$ is $+1$ when a successful trajectory demonstrates the atom's procedure or a failed trajectory identifies it as a relevant correction. It is -1 when the instruction was followed and contributed to failure, and zero when the atom is unassessed or the evidence is unclear.

The optimizer also uses a^{eff} as a reference for identifying redundant reasoning. For an example executed before, output-length changes help locate possible inefficiency. The optimizer then examines the trajectory for behaviors such as repeating a check without new information and links them to the instructions responsible. An implicated accuracy atom receives efficiency evidence $v_{i,t}^{\mathrm{eff}}(x) = -1$, even if the task succeeds, and otherwise zero. Both sources assign negative evidence only to atoms included in the executed skill. We combine them with negative evidence taking precedence:

$$\varepsilon_{i,t}(x) = \begin{cases} -1, & v_{i,t}^{\mathrm{acc}}(x) = -1 \text{ or } v_{i,t}^{\mathrm{eff}}(x) = -1, \\ v_{i,t}^{\mathrm{acc}}(x), & \text{otherwise.} \end{cases} \tag{4}$$

Evidence accumulates across distinct examples, with each example counted once per atom. Negative evidence permanently replaces earlier positive support from the same example. Let $p_{i,t}$ and $n_{i,t}$ count the examples providing positive and negative evidence for atom a_i before step t, respectively.

Activation also checks *specificity* q_i, which measures how explicitly the atom states when it applies. We compute it from the presence of a problem category and capped counts of relevance cues, required assumptions, and excluded situations. A confidence score $c_{i,t}$ combines specificity with positive and negative support. Appendix A.4 gives both calculations and the activation thresholds $q_{\min}$ and $c_{\min}$. After batch t, candidates become *active* (eligible for inclusion in the skill), and active atoms become *retired* (permanently removed from use), under the following rules:

$$\text{candidate} \xrightarrow[q_i \geq q_{\min},\ c_{i,t+1} \geq c_{\min}]{p_{i,t+1} \geq 2,\ n_{i,t+1} = 0} \text{active} \xrightarrow{n_{i,t+1} \geq 2} \text{retired}, \tag{5}$$

with each atom retaining its current state otherwise. These decisions reuse $\mathcal{F}_t$ without candidate-validation rollouts, giving $C_t^{\mathrm{gate}} = 0$ in Eq. (3).

3.4 Skill Composition

To obtain the final skill, FreeEvo combines the fixed efficiency atom with the active accuracy atoms. It first reserves space for a^{eff} within a skill token budget B. Active accuracy atoms are then ranked by their confidence score and specificity. They are considered in that order, including each atom that fits in the remaining budget. Appendix A.5 specifies the ranking and token estimates. An atom omitted by the budget remains active and can be included in a later composition.

The same rule composes skills between evolution batches. Let $\mathcal{A}_t$ contain the discovered accuracy atoms and $\mathcal{I}_t$ be the ordered indices selected by this rule for step t. For $t \geq 1$, the composed skill is

$$s_t = \mathrm{Concat}\big(a^{\mathrm{eff}},\ (a_i)_{i \in \mathcal{I}_t}\big), \tag{6}$$

where Concat places the selected instructions and their conditions in the agent's prompt. During evolution, the composed skill guides the next batch, whose feedback updates atom selection. After T evolution steps, FreeEvo returns s_T and keeps it fixed for testing.

Table 1: **Task performance and token cost across three models.** Evo: evolution input plus completion tokens (M); Test: full-test output tokens (M). FreeEvo changes use No Skill for performance/Test and SkillOpt for Evo as references.

	Olympiad			GPQA-D			TriviaQA			ALFWorld		
Method	Acc↑	Evo↓	Test↓	Acc↑	Evo↓	Test↓	EM↑	Evo↓	Test↓	SR↑	Evo↓	Test↓
Qwen3.6-35B-A3B												
No Skill	82.2	0.00	11.80	81.2	0.00	0.86	84.0	0.00	0.94	61.2	0.00	5.16
GRASP	79.3	48.96	**6.16**	81.9	6.62	0.63	**85.9**	93.78	0.90	79.1	35.51	2.86
GRASP w/o Gate	78.3	15.66	7.51	75.4	2.87	0.64	85.1	29.00	0.95	75.4	9.06	2.86
SkillOpt	82.5	56.81	10.90	81.2	5.25	0.69	85.2	175.27	0.85	82.1	19.57	1.98
SkillOpt w/o Gate	80.8	31.23	11.49	81.2	4.60	0.72	84.8	67.42	0.89	80.7	14.41	2.17
FreeEvo	**84.6** (+2.4)	**13.12** (-76.9%)	10.03 (-15.0%)	**82.6** (+1.4)	**1.41** (-73.1%)	**0.62** (-27.9%)	**85.9** (+1.9)	**11.22** (-93.6%)	**0.84** (-10.6%)	**84.3** (+23.1)	**6.58** (-66.4%)	**1.14** (-77.9%)
DeepSeek V4 Flash												
No Skill	71.5	0.00	7.40	87.7	0.00	1.30	85.7	0.00	0.36	57.5	0.00	1.13
GRASP	77.6	54.94	7.12	91.3	10.36	**1.10**	85.6	89.24	0.53	50.8	25.75	1.08
GRASP w/o Gate	70.8	13.92	7.27	89.9	3.85	1.15	84.2	26.13	0.54	48.3	8.29	1.49
SkillOpt	73.8	40.72	6.90	87.0	8.90	1.29	89.0	191.26	0.43	62.7	14.59	0.93
SkillOpt w/o Gate	73.4	29.24	7.18	86.3	7.45	1.37	88.1	83.70	0.48	60.4	17.66	0.97
FreeEvo	**77.9** (+6.4)	**10.37** (-74.5%)	**6.48** (-12.4%)	**91.8** (+4.1)	**2.57** (-71.1%)	1.22 (-6.2%)	**89.6** (+3.9)	**16.34** (-91.5%)	**0.35** (-2.8%)	**64.4** (+6.9)	**5.83** (-60.0%)	**0.55** (-51.3%)
GLM 4.7 Flash												
No Skill	65.8	0.00	7.92	66.7	0.00	1.50	74.6	0.00	1.81	41.8	0.00	5.57
GRASP	68.8	53.02	6.05	63.7	11.35	1.34	76.2	104.07	1.98	60.0	46.21	4.17
GRASP w/o Gate	64.8	19.09	6.75	59.8	4.56	1.36	75.0	31.32	2.06	56.9	13.24	4.90
SkillOpt	68.7	48.80	8.66	59.4	11.70	1.58	**79.0**	213.70	2.20	73.2	33.46	3.23
SkillOpt w/o Gate	68.5	29.36	8.72	57.5	10.11	1.61	76.0	98.25	2.34	72.4	25.17	3.53
FreeEvo	**70.4** (+4.6)	**18.25** (-62.6%)	**5.99** (-24.4%)	**67.0** (+0.3)	**3.11** (-73.4%)	**1.32** (-12.0%)	78.8 (+4.2)	**18.73** (-91.2%)	**1.75** (-3.3%)	**74.1** (+32.3)	**12.51** (-62.6%)	**1.88** (-66.2%)

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Tasks and models. We evaluate Qwen3.6-35B-A3B (Yang et al., 2025), DeepSeek V4 Flash (Xu et al., 2026), and GLM 4.7 Flash (Zeng et al., 2026) on OlympiadBench (He et al., 2024) (mathematical reasoning), GPQA-Diamond (Rein et al., 2023) (science QA), TriviaQA (Joshi et al., 2017) (reading comprehension), and ALFWorld (Shridhar et al., 2020) (interactive tasks). The execution model remains frozen. Appendix C provides data splits and model configurations.

Baselines and protocol. We compare with No Skill, GRASP (Moll et al., 2026), SkillOpt (Yang et al., 2026), and their gate-free variants under matched prompts, splits, scoring, execution settings, training coverage, and update schedules. Gate-free baselines deploy their final skills without validation-based acceptance or selection; GRASP also disables candidate probes and collapse recovery (Appendix C.2). Performance and token usage average four evaluations unless noted otherwise.

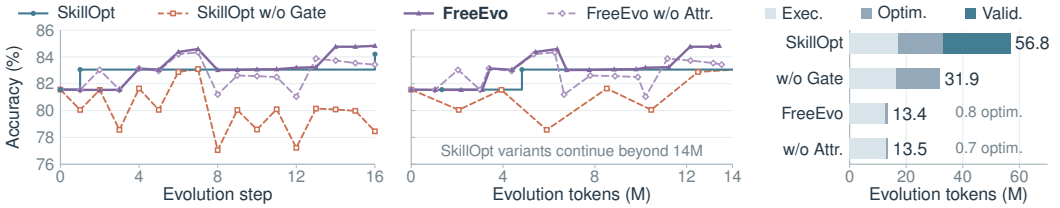
Metrics. Performance is accuracy (Olympiad and GPQA-D), exact match (EM; TriviaQA), or success rate (SR; ALFWorld), in percent. Evo counts evolution input and completion tokens, including optimizer and validation calls; Test counts full-test output tokens, including all ALFWorld turns. Both use millions (M). Performance changes use percentage points; token changes use percentages. Bold/underlined values mark the best/second-best distinct results, including ties; No Skill is excluded from Evo rankings.

4.2 MAIN RESULTS

In Tab. 1, FreeEvo improves performance and reduces Test relative to No Skill in all twelve settings, averaging +7.6 points and -25.8%. Relative to SkillOpt, it gains 2.3 points while using 25.2% of Evo. Averages weight each setting's performance difference or token ratio equally. FreeEvo

Table 2: **Component ablations on Qwen3.6-35B-A3B.** Changes are relative to full FreeEvo; metrics follow Tab. 1.

	Olympiad			GPQA-D			TriviaQA		
Variant	Acc↑	Evo (M)↓	Test (M)↓	Acc↑	Evo (M)↓	Test (M)↓	EM↑	Evo (M)↓	Test (M)↓
Full FreeEvo	84.6 (+0.0)	**13.12** (+0.0%)	**10.03** (+0.0%)	**82.6** (+0.0)	1.41 (+0.0%)	**0.62** (+0.0%)	**85.9** (+0.0)	**11.22** (+0.0%)	**0.84** (+0.0%)
w/o fixed guidance	83.7 (-0.9)	13.98 (+6.6%)	10.80 (+7.7%)	81.3 (-1.3)	**1.40** (-0.7%)	0.69 (+11.3%)	85.6 (-0.3)	11.42 (+1.8%)	0.93 (+10.7%)
w/o efficiency attribution	83.1 (-1.5)	13.57 (+3.4%)	11.01 (+9.8%)	81.9 (-0.7)	1.42 (+0.7%)	0.68 (+9.7%)	85.0 (-0.9)	11.29 (+0.6%)	0.93 (+10.7%)
w/o both efficiency paths	82.4 (-2.2)	14.26 (+8.7%)	11.55 (+15.2%)	79.3 (-3.3)	1.43 (+1.4%)	0.75 (+21.0%)	84.9 (-1.0)	11.50 (+2.5%)	0.98 (+16.7%)
w/o retirement	**84.7** (+0.1)	13.14 (+0.2%)	10.22 (+1.9%)	81.9 (-0.7)	1.41 (+0.0%)	0.66 (+6.5%)	85.6 (-0.3)	11.23 (+0.1%)	0.89 (+6.0%)



(a) Evolution dynamics (b) Evolution cost

Figure 5: **Evolution on Olympiad with Qwen3.6-35B-A3B.** (a) Checkpoint accuracy by update step and cumulative tokens; both SkillOpt variants continue beyond the 14M window. (b) Full-run token costs (M), split into execution, optimization, and validation.

achieves best or tied-best performance in eleven settings and the lowest Evo among skill-based methods in all twelve.

On GLM TriviaQA, FreeEvo trails SkillOpt by 0.2 points but saves 91.2% Evo and 20.5% Test. Removing the gate reduces GRASP and SkillOpt performance by 3.0 and 1.1 points on average; FreeEvo outperforms both ungated variants in every setting.

4.3 Component Ablations

Tab. 2 removes the fixed efficiency atom, disables efficiency attribution ($v_{i,t}^{\text{eff}}(x) = 0$), or removes both. A fourth variant disables retirement while preserving evidence accumulation and activation.

On Olympiad, GPQA-D, and TriviaQA, efficiency attribution improves performance by 1.5, 0.7, and 0.9 points and reduces Test by 8.9%, 8.8%, and 9.7% under the same fixed guidance. Removing guidance alone lowers performance by 0.9, 1.3, and 0.3 points; removing both paths increases these losses to 2.2, 3.3, and 1.0 points. Disabling retirement changes performance by +0.1, -0.7, and -0.3 points while increasing Test on all three tasks.

4.4 Evolution Dynamics

In Fig. 5, FreeEvo reaches 84.8% checkpoint accuracy after sixteen updates, versus 83.4% without efficiency attribution and 78.5% for SkillOpt w/o Gate. Their maximum drops from a preceding peak are 1.6, 3.3, and 6.0 points, respectively. Within 14M tokens, FreeEvo completes its run while SkillOpt's last completed checkpoint reaches 83.1%. Validation accounts for 42.0% of SkillOpt's evolution cost; FreeEvo uses 23.6% of its total, including optimizer processing.

4.5 Cross-Model Transfer

We apply frozen Qwen skills to DeepSeek without further evolution or editing (Tab. 3). Across Olympiad, GPQA-D, and TriviaQA, transferred FreeEvo improves performance over No Skill by 2.6 points on average and outperforms transferred SkillOpt on every task, but trails native FreeEvo

Table 3: **Cross-model transfer from Qwen3.6-35B-A3B to DeepSeek V4 Flash.** Evo is source evolution cost (M); Test is target output cost (M). Performance/Test changes use DeepSeek No Skill; Evo changes use Qwen SkillOpt.

	Olympiad			GPQA-D			TriviaQA		
Method	Acc↑	Src. Evo↓	Tgt. Test↓	Acc↑	Src. Evo↓	Tgt. Test↓	EM↑	Src. Evo↓	Tgt. Test↓
No Skill	71.5 (+0.0)	0.00 (+0.0%)	7.40 (+0.0%)	87.7 (+0.0)	0.00 (+0.0%)	1.30 (+0.0%)	85.7 (+0.0)	0.00 (+0.0%)	0.36 (+0.0%)
SkillOpt	72.9 (+1.4)	56.81 (+0.0%)	7.28 (-1.6%)	88.4 (+0.7)	5.25 (+0.0%)	1.38 (+6.2%)	86.9 (+1.2)	175.27 (+0.0%)	0.63 (+75.0%)
FreeEvo	75.5 (+4.0)	13.12 (-76.9%)	7.39 (-0.1%)	89.1 (+1.4)	**1.41** (-73.1%)	1.37 (+5.4%)	88.1 (+2.4)	**11.22** (-93.6%)	0.58 (+61.1%)
Native reference: evolved on DeepSeek V4 Flash									
FreeEvo (native)†	**77.9** (+6.4)	**10.37**† (-81.7%)	**6.48** (-12.4%)	**91.8** (+4.1)	2.57† (-51.0%)	**1.22** (-6.2%)	**89.6** (+3.9)	16.34† (-90.7%)	**0.35** (-2.8%)

† Native DeepSeek results from Tab. 1; marked Evo costs are incurred on DeepSeek.

Table 4: **Cross-dataset transfer on Qwen3.6-35B-A3B.** Frozen source skills are reused across target benchmarks. Changes use target No Skill for performance/Test and same-source SkillOpt for Evo. Evo is ranked within each source.

		AIME25		AIME26		MATH-500		Olympiad	
Method	Src. Evo (M)↓	Acc↑	Test (M)↓	Acc↑	Test (M)↓	Acc↑	Test (M)↓	Acc↑	Test (M)↓
No Skill	0.00 (+0.0%)	86.8 (+0.0)	0.98 (+0.0%)	90.3 (+0.0)	0.95 (+0.0%)	97.8 (+0.0)	5.98 (+0.0%)	82.2 (+0.0)	11.80 (+0.0%)
Source: Olympiad									
SkillOpt	56.81 (+0.0%)	91.2 (+4.4)	1.05 (+7.1%)	92.0 (+1.7)	1.04 (+9.5%)	98.8 (+1.0)	4.58 (-23.4%)	82.5 (+0.3)	10.90 (-7.6%)
FreeEvo	**13.12** (-76.9%)	**92.3** (+5.5)	0.87 (-11.2%)	**93.2** (+2.9)	0.91 (-4.2%)	**99.0** (+1.2)	4.56 (-23.7%)	**84.6** (+2.4)	10.03 (-15.0%)
Source: MATH									
SkillOpt	78.69 (+0.0%)	89.7 (+2.9)	1.00 (+2.0%)	88.9 (-1.4)	1.01 (+6.3%)	98.2 (+0.4)	5.64 (-5.7%)	81.6 (-0.6)	10.66 (-9.7%)
FreeEvo	**13.22** (-83.2%)	90.2 (+3.4)	**0.80** (-18.4%)	93.1 (+2.8)	**0.83** (-12.6%)	98.2 (+0.4)	**3.14** (-47.5%)	82.9 (+0.7)	**8.00** (-32.2%)

by 2.2 points. Test increases by 22.1% on average, driven mainly by TriviaQA (+61.1%). Thus, performance gains transfer more consistently than output-token savings.

4.6 Cross-Dataset Transfer

On Qwen, we transfer an Olympiad skill to AIME25, AIME26, and MATH-500, and a skill evolved on the MATH training split to these targets plus Olympiad (Tab. 4). Excluding the in-domain Olympiad reference, FreeEvo gains 3.2 and 1.8 points over No Skill and saves 13.1% and 27.7% Test on average for the Olympiad and MATH sources, respectively. All seven transfers improve performance and reduce Test, while matching or exceeding SkillOpt's accuracy at lower Evo and Test costs. Olympiad skills achieve higher accuracy on the three shared targets; MATH skills use fewer output tokens.

5 Conclusion

Validation gates stabilize skill self-evolution at substantial cost, while harmful updates often manifest as inefficient reasoning. Motivated by this observation, **FreeEvo** combines a fixed efficiency atom with evidence across execution trajectories to activate useful accuracy atoms and permanently retire harmful ones without candidate-validation rollouts. Across three models and four task domains, it improves performance over SkillOpt by 2.3 percentage points with 74.8% fewer evolution tokens on average. Ablations show that efficiency attribution improves performance and output efficiency beyond fixed guidance alone. Frozen skills improve performance across datasets and models, while token savings transfer more consistently across datasets. These findings support reasoning efficiency as feedback for instruction selection, enabling cheaper skill self-evolution.

AI USE STATEMENT

We use generative AI tools to edit the manuscript language for readability, to create or modify scientific figures and illustrative visual assets, and to assist with implementing our methods. All AI-suggested language edits are reviewed and revised by the authors, every figure is verified against the underlying experimental data, results, and intended content, and all AI-assisted code is reviewed, tested, and verified by the authors before use. We do not use generative AI tools to design the research, formulate hypotheses or mathematical claims, write proofs, generate datasets, clean or reformat data, conduct qualitative analysis, or interpret results. The authors take full responsibility for the final content of this work.

REFERENCES

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

Kunfeng Chen, Qihuang Zhong, Juhua Liu, and Bo Du. Skillcat: Contrastive assessment and topology-aware skill self-evolution for llm agents. *arXiv preprint arXiv:2606.13317*, 2026.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Constructing instruction manuals by llm agents via interactive environmental learning. *Advances in Neural Information Processing Systems*, 37:589–631, 2024a.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*, 2024b.

Jingzhi Gong, Ruizhen Gu, Zhiwei Fei, Yazhuo Cao, Lukas Twist, Alina Geiger, Shuo Han, Dominik Sobania, Federica Sarro, and Jie M Zhang. Skillmoo: Multi-objective optimization of agent skills for software engineering. *arXiv preprint arXiv:2604.09297*, 2026.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

Yu He and Weikai Yang. Skillcommit: Evolving agent skills through behaviorally validated scope expansion. *arXiv preprint arXiv:2608.15165*, 2026.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Gaoyuan Li, Meihao Fan, Yizhe Liu, Shaolei Zhang, Ju Fan, Siyi Wang, Jiaheng Hou, Xudong Weng, Honghan Tian, and Zang Li. Skilladam: Stable and efficient skill evolution for agents. *arXiv preprint arXiv:2609.08944*, 2026a.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026b.

Huawei Lin, Peng Li, Jie Song, Fuxin Jiang, and Tieying Zhang. Muse-autoskill: Self-evolving agents via skill creation, memory, management, and evaluation. *arXiv preprint arXiv:2605.27366*, 2026.

Yuxuan Liu, Zhaochen Su, Yuhao Zhang, Jiahe Guo, Zhongwei Xie, Huihao Jing, Lingyun Xie, Qing Zong, Yauwai Yim, Zhixiong Zhang, et al. Rethinking self-evolving agent skills: Feedback dynamics over multiple rounds. *arXiv preprint arXiv:2608.02636*, 2026.

Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. Skillgen: Verified inference-time agent skill synthesis. *arXiv preprint arXiv:2605.10999*, 2026a.

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. Skillclaw: Let skills evolve collectively with agentic evolver. *arXiv preprint arXiv:2604.08377*, 2026b.

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.

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*, volume 2026, pp. 94327–94354, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Linfang Shang, Ming Xu, Yiding Sun, Tianle Xia, Lingxiang Hu, Lan Xu, and Ning Zheng. When self-evolution backfires: Pre-commit gating against skill contamination in llm agents. *arXiv preprint arXiv:2608.05810*, 2026.

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

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.

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

Hanyu Wang, Yifan Lan, Bochuan Cao, Lu Lin, and Jinghui Chen. Skillgrad: Optimizing agent skills like gradient descent. *arXiv preprint arXiv:2605.27760*, 2026a.

Xuhui Wang, Ruoqi Shu, Chen Dan, Tianhua Xu, Mengxi Luo, Yanming Mai, and Bo Wan. Frames: Guarded and dual-objective skill evolution for agents in policy-governed enterprise workflows. *arXiv preprint arXiv:2608.01772*, 2026b.

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

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, et al. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 7920–7939, 2025.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

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

Ye Yu, Xiaopeng Yuan, Haibo Jin, Heming Liu, Yaoning Yu, and Haohan Wang. Do self-evolving agents forget? capability degradation and preservation in lifelong llm agent adaptation. *arXiv preprint arXiv:2605.09315*, 2026.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

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*, volume 2026, pp. 86069–86100, 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.

Guangxiang Zhao, Qilong Shi, Xusen Xiao, Xiangzheng Zhang, Tong Yang, and Lin Sun. Thinking with reasoning skills: Fewer tokens, more accuracy. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL 2026)*, pp. 2295–2308, 2026.

CONTENTS

A Method Details **14**

A.1 Evolution Procedure . . . 14
A.2 Atom Representation and Discovery . . . 14
A.3 Trajectory Evidence Attribution . . . 15
A.4 Evidence Accumulation and Atom Scores . . . 15
A.5 Skill Composition . . . 16
A.6 Illustrative Example: Checking Extraneous Roots . . . 16

B Related Work **17**

B.1 Reliable and Efficient Skill Self-Evolution . . . 18
B.2 Reasoning Efficiency and Cost-Aware Skill Adaptation . . . 18
B.3 Fine-Grained Experience Management . . . 19

C Experimental Details **19**

C.1 Data and Execution Settings . . . 20
C.2 Baseline Settings . . . 21
C.3 Token Accounting and Repeated Evaluation . . . 21
C.4 Transfer Protocols . . . 21
C.5 Fixed Efficiency Guidance . . . 22

D FreeEvo Prompt Templates **23**

D.1 Efficiency Atom Initialization . . . 23
D.2 Accuracy Atom Discovery . . . 24
D.3 Task-Outcome Evidence Attribution . . . 25
D.4 Efficiency Evidence Attribution . . . 26
D.5 Skill Composition and Task Execution . . . 27

A METHOD DETAILS

A.1 EVOLUTION PROCEDURE

Each evolution step uses one batch of execution feedback to build the skill for the next batch. Alg. 1 organizes this process into four stages: execution, atom generation, evidence attribution, and skill selection. The agent π_θ executes the tasks; the optimizer $\mathcal{U}$ proposes atoms and assesses their effects on the resulting trajectories. This evidence determines which accuracy atoms become active or retire. The next skill combines the fixed efficiency atom with active accuracy atoms that fit the budget, following Appendix A.4 and Appendix A.5.

Before step t, $\mathcal{A}_t$ stores the discovered accuracy atoms, their evidence, and their states; $\mathcal{I}_t$ lists the ordered indices of those included in s_t. The feedback history $\mathcal{H}_t$ stores earlier trajectories, outcomes, and output lengths. It supports discovery across batches and comparisons when examples recur. Initially, these collections are empty, and the first batch uses only the standard task prompt. After T steps, the final skill s_T is frozen for testing.

Algorithm 1 FreeEvo: gate-free skill self-evolution

Input: Batches $\{\mathcal{B}_t\}_{t=0}^{T-1}$, agent π_θ, optimizer $\mathcal{U}$, skill token budget B.
Initialize: $\mathcal{A}_0 = \emptyset$, $\mathcal{I}_0 = \emptyset$, $\mathcal{H}_0 = \emptyset$, $s_0 = \emptyset$.
For $t = 0, \ldots, T-1$:

1. **Execute the current skill.**
   Execute $\mathcal{B}_t$ with s_t; collect trajectories, outcomes, and lengths as $\mathcal{F}_t$.
2. **Generate or refine atoms.**
   If $t = 0$: generate a^{eff} from $\mathcal{F}_0$ and keep it fixed thereafter.
   If in the first epoch: discover accuracy atoms from recurring patterns in $\mathcal{H}_t \cup \mathcal{F}_t$.
   Initialize each new atom as a candidate with empty support sets.
   Otherwise: refine only existing accuracy atoms with no evidence.
3. **Attribute and accumulate evidence.**
   Assess task-outcome evidence $v_{i,t}^{\text{acc}}(x)$ from $\mathcal{F}_t$.
   Use a^{eff} and the history $\mathcal{H}_t$ to identify redundant reasoning in $\mathcal{F}_t$.
   Set $v_{i,t}^{\text{eff}}(x) = -1$ for implicated atoms in $\mathcal{I}_t$, and zero otherwise.
   Restrict negative evidence from either source to atoms used in s_t.
   Combine the judgments, giving negatives precedence (Eq. (4)).
   Update support sets over distinct examples (Eq. (8)).
   Freeze the instruction and conditions of every atom that has received evidence.
4. **Select atoms and compose the next skill.**
   Compute confidence scores (Eq. (10)).
   Update candidate, active, and retired states to obtain $\mathcal{A}_{t+1}$ (Eq. (5)).
   Reserve space for a^{eff} within B; rank active accuracy atoms.
   Select those that fit in the remaining budget, in order, to obtain $\mathcal{I}_{t+1}$.
   Compose s_{t+1} from a^{eff} and the selected atoms (Eq. (6)).
   Store the batch feedback: $\mathcal{H}_{t+1} = \mathcal{H}_t \cup \mathcal{F}_t$.

End for
Return: The frozen final skill s_T.

A.2 ATOM REPRESENTATION AND DISCOVERY

Procedures and application conditions. An accuracy atom $a_i = (u_i, \mathcal{C}_i)$ records a reusable procedure and the situations in which it applies. For example, u_i may instruct the agent to substitute candidate roots into the original equation, with $\mathcal{C}_i$ specifying that an earlier transformation could have introduced extraneous roots. We record the conditions in four parts: a problem category, relevance cues, required assumptions, and excluded situations. The root-checking atom belongs to equation solving, is relevant after a transformation such as squaring, and assumes the original equation is available for substitution. This structure gives the optimizer a concrete instruction and scope against which to assess trajectories.

Initialization. The first batch uses the standard task prompt with no evolved skill, so $s_0 = \emptyset$. The optimizer uses its feedback to generate the efficiency atom a^{eff}. The atom expresses task-specific guidance through concrete behaviors, such as proceeding once a check has resolved the relevant uncertainty. It is fixed and included from step 1 onward, providing the same reference for execution and subsequent efficiency attribution.

Accuracy atom discovery. New accuracy atoms are generated only during the first epoch, from its first batch through its last. The optimizer proposes them from recurring patterns across multiple distinct examples in the stored feedback. It compares each proposal with existing atoms to specify a distinct procedural role and applicable situation. Each new atom enters as a candidate with empty support sets and is then assessed against the analyzed trajectories. A successful trajectory can demonstrate the proposed procedure, while analysis of a failure can identify it as a relevant correction. Such evidence allows a candidate to accumulate support before activation.

Once an atom receives evidence, its instruction and conditions are frozen so that future judgments refer to the same content. After the first epoch, no new accuracy atoms are added. Evolution continues to accumulate evidence and update atom selection, with refinements confined to existing atoms that have not received evidence. The fixed efficiency atom is outside this discovery and selection process after initialization.

A.3 Trajectory Evidence Attribution

Task-outcome evidence. For each assessed atom on example x, the optimizer assigns $v_{i,t}^{\text{acc}}(x) \in \{-1, 0, +1\}$. Positive evidence indicates either a useful procedure demonstrated in a successful trajectory or a relevant correction identified from a failed trajectory. Negative evidence requires that the atom was included in s_t, followed during execution, and implicated in task failure. Zero denotes an unassessed atom or a trajectory that provides no sufficiently clear evidence. Every nonzero judgment includes an explanation identifying the instruction and the behavior supporting the assignment.

Efficiency evidence. For a revisited example, let $t^-(x)$ be its previous execution step. The output-length change is

$$\Delta \ell_{x,t} = \ell_{x,t} - \ell_{x,t^-(x)}. \tag{7}$$

An increase directs attention to possible additional exploration. The optimizer uses a^{eff} to examine what that exploration accomplishes and identify redundant behavior. Examples include repeating a check without new information, revisiting an unchanged failed approach, or repeatedly switching between conflicting instructions. For an example without a prior execution, the optimizer assesses the current trajectory against the efficiency guidance.

If the optimizer links a redundant behavior to an accuracy atom included in s_t, it assigns $v_{i,t}^{\text{eff}}(x) = -1$. Otherwise, it assigns zero. Newly activated atoms receive particular attention because they changed the guidance available to the agent. Each negative assignment identifies both the instruction and the redundant behavior attributed to it, including when the task ultimately succeeds.

Combining judgments. Eq. (4) combines the two sources and gives negative evidence precedence. For example, a checking procedure may help obtain the correct answer while inducing repeated verification after the uncertainty is resolved. Attributing that repetition to the procedure makes the combined evidence negative. Both sources restrict negative attribution to atoms actually included in the executed skill, so an active atom omitted by the length budget cannot receive negative evidence from that execution.

A.4 Evidence Accumulation and Atom Scores

Distinct-example support. For each atom, let $\mathcal{P}_{i,t}$ and $\mathcal{N}_{i,t}$ contain the examples providing positive and negative evidence before step t. Both sets are initially empty. After analyzing the current batch, we update them as

$$\begin{aligned} \mathcal{N}_{i,t+1} &= \mathcal{N}_{i,t} \cup \{x \in \mathcal{B}_t : \varepsilon_{i,t}(x) = -1\}, \\ \mathcal{P}_{i,t+1} &= (\mathcal{P}_{i,t} \cup \{x \in \mathcal{B}_t : \varepsilon_{i,t}(x) = +1\}) \setminus \mathcal{N}_{i,t+1}. \end{aligned} \tag{8}$$

The counts used in Sec. 3.3 are $p_{i,t} = |\mathcal{P}_{i,t}|$ and $n_{i,t} = |\mathcal{N}_{i,t}|$. Set membership counts each example once per atom across epochs. If both analyses assign negative evidence to the same atom on an example, the negative set receives one member. A negative assignment removes earlier positive support from that example and remains recorded in subsequent epochs.

Specificity of application conditions. Let $b_i \in \{0, 1\}$ indicate whether a problem category is specified, and let T_i, P_i, and X_i contain the relevance cues, required assumptions, and excluded situations, respectively. We compute

$$q_i = \min\left(1, \frac{b_i + \min(|T_i|, 3) + \min(|P_i|, 2) + \min(|X_i|, 2)}{6}\right). \tag{9}$$

The capped counts measure how explicitly the atom states its application conditions while limiting the contribution of any one category.

Confidence and state transitions. The atom confidence score combines the balance of supporting and opposing examples with specificity:

$$c_{i,t} = \underbrace{\frac{p_{i,t} + \alpha}{p_{i,t} + n_{i,t} + \alpha + \beta}}_{\text{smoothed support ratio}} \underbrace{\frac{1 + \max(0.1, q_i)}{2}}_{\text{specificity weight}}. \tag{10}$$

We use $\alpha = \beta = 1$, $q_{\min} = 0.35$, and $c_{\min} = 0.55$. Eq. (5) activates a candidate only when it has at least two distinct positive examples, no negative evidence, and meets both the specificity and confidence thresholds. An active atom retires after two distinct negative examples. Otherwise, the current state is retained. Retired atoms remain retired. These rules govern accuracy atoms, while a^{eff} remains fixed.

A.5 Skill Composition

Composition first reserves space for a^{eff} within the estimated skill token budget B. Active accuracy atoms are ordered by decreasing $c_{i,t}$, decreasing q_i, increasing $n_{i,t}$, increasing estimated token cost, and finally atom identifier to resolve ties. Each atom that fits within the remaining budget is included with its instruction and conditions. The estimate uses one token per four characters, rounded up, for the instruction, relevance cues, assumptions, and excluded situations. We use $B = 1200$ for MATH, OlympiadBench, GPQA-D, and ALFWorld, and $B = 1800$ for TriviaQA. An atom excluded by the budget retains its state and remains eligible for later selection. Appendix D provides the prompt templates for atom generation, evidence attribution, skill composition, and task execution.

A.6 Illustrative Example: Checking Extraneous Roots

Fig. 6 illustrates the complementary roles of accuracy and efficiency atoms using the equation $\sqrt{x+2} = x$. The cards reproduce example atom instructions and selected application conditions, while the solution below is a constructed illustration. The accuracy atom specifies a reusable verification procedure and its scope. The efficiency atom describes when to reuse completed reasoning and when new evidence warrants further work.

Squaring produces $x^2 - x - 2 = 0$, whose roots are 2 and -1. The accuracy atom directs the solver to check both against the original equation: $\sqrt{2+2} = 2$, whereas $\sqrt{-1+2} = 1 \neq -1$. This establishes the complete solution $x = 2$. At this point, factoring the same polynomial again or repeating the same substitutions adds no information. The efficiency atom calls for concluding the solution, while allowing reconsideration if new evidence identifies a specific gap. Together, the instructions preserve the check that removes the extraneous root and identify when its purpose has been fulfilled.

Accuracy atom

Do: Substitute every surviving candidate into the **original condition** and check domain, codomain, sign, denominator, and substitution constraints.

Use when: candidates from a reduced equation; domain-changing substitution.

Requires: a candidate set has been derived.

Do not use when: the original condition was **already checked**.

Efficiency atom

Do: Maintain only concrete unresolved proof obligations. **Reuse settled arguments and calculations**; do not repeat unchanged work or an unchanged failed attempt. Reconsider a settled step only when **new evidence identifies a specific gap**. Once every obligation is resolved, give a concise complete proof using the existing answer-format rules and end.

Use when: reasoning would repeat settled work, retry an unchanged failed branch, or continue after all requested obligations are resolved.

Requires: preserve necessity, sufficiency, cases, domain, boundary, construction, and candidate checks.

Do not use when: new evidence requires reconsideration to close a named gap or contradiction; efficiency never licenses omitting proof or checks.

Illustrative application *Constructed example*

$$\sqrt{x+2} = x \xrightarrow{\text{square}} x^2 - x - 2 = 0 \implies x \in \{2, -1\}$$

Apply the accuracy atom

Verify each candidate in the original equation

$x = 2$: $\sqrt{2+2} = 2$ — **Keep** $x = 2$

$x = -1$: $\sqrt{-1+2} = 1 \neq -1$ — **Reject the extraneous root**

All checks complete

No new evidence

Efficiency atom: conclude

Final answer: $\boxed{x = 2}$

Efficiency atom: skip repetition

Factor the same polynomial again
Substitute the same candidates again

Figure 6: **Accuracy and efficiency atoms in an illustrative root-checking example.** Squaring yields two candidates; substitution into the original equation retains 2 and rejects the extraneous root -1. After these checks, the green branch concludes the solution. The dashed orange branch depicts redundant calculations discouraged by the efficiency atom. The necessary verification is completed before termination.

B Related Work

Language model agents can retain lessons from execution as external guidance for subsequent tasks. Reflexion (Shinn et al., 2023) stores verbal reflections to improve later attempts, while ExpeL (Zhao et al., 2024) extracts reusable insights from successful and failed experiences. Recent skill evolution frameworks extend this approach to persistent skill repositories. SkillClaw (Ma et al., 2026b) aggregates experience across users and sessions, and EvoSkill (Alzubi et al., 2026) discovers and

refines skills through execution feedback. These approaches make accumulated experience available beyond its original interaction. Their effectiveness depends on which instructions are retained, how they influence subsequent execution, and how much computation their revision requires.

B.1 Reliable and Efficient Skill Self-Evolution

Repeated revision does not ensure continuous improvement. Rethinking Self-Evolving Agent Skills (Liu et al., 2026) characterizes multi-round evolution as a sparse search in which only some candidates become new validation-best versions. Studies of lifelong adaptation also identify capability erosion as agents adapt to new task distributions (Yu et al., 2026). When Self-Evolution Backfires (Shang et al., 2026) examines a related mechanism in skill pools: defective instructions can influence the distillation of later skills, allowing errors to propagate across rounds. Together, these findings motivate update rules that assess the consequences of new guidance and preserve useful behavior.

Candidate execution provides a direct basis for these decisions. SkillOpt (Yang et al., 2026) accepts bounded edits when they strictly improve performance on a held-out selection split. SkillOpt-Lite (Shen et al., 2026) simplifies the revision pipeline while retaining independent validation gating. SkillGen (Ma et al., 2026a) uses a construction-time verification subset to measure repairs and regressions, refine candidates, and select a skill with positive net benefit. GRASP (Moll et al., 2026) evaluates proposed library changes on probes drawn from historical development episodes, requiring a net gain under a regression constraint. These procedures protect observed task performance by executing candidates before adoption, adding rollouts to the evolution process.

Other approaches control the revision process through accumulated feedback. SkillAdam (Li et al., 2026a) maintains an optimization memory and adapts the edit budget to variation in case-level improvements. Its acceptance decision still uses candidate executions on the same mini-batch that produced the revision. SkillGrad (Wang et al., 2026a) instead combines trajectory diagnoses with textual momentum and directly applies structured patches, without validation-based selection of each update. These designs illustrate how both the source of feedback and the mechanism that acts on it shape evolution.

FreeEvo uses task outcomes and reasoning-efficiency evidence from the ongoing evolution loop to determine which instructions enter or leave the skill. The optimizer examines how an instruction contributes to successful procedures, errors, or redundant exploration, and evidence accumulated across examples controls its continued use. This shifts the adoption decision to instruction-level evidence from existing trajectories, removing the additional candidate executions required by performance gates. Reasoning efficiency supplies a further signal for identifying harmful guidance beyond whether a task is ultimately solved.

B.2 Reasoning Efficiency and Cost-Aware Skill Adaptation

The efficiency of a skill depends on the reasoning behavior it induces. Work on overthinking (Chen et al., 2024b) identifies repeated exploration and verification that consume computation with limited benefit to the final answer. Thinking with Reasoning Skills (Zhao et al., 2026) shows how reusable procedures can address this problem. It distills skills from reasoning experience and retrieves relevant guidance for new problems, reducing reasoning tokens while improving performance on mathematical and coding tasks. This connects procedural knowledge to more direct solution paths and makes execution behavior a useful object of analysis when updating skills.

Recent methods explicitly incorporate cost into skill optimization. SkillMOO (Gong et al., 2026) evolves skill bundles using Pareto selection over pass rate and inference cost. FRAMES (Wang et al., 2026b) combines a Pareto frontier over accuracy and cost with category-level regression constraints and checks against historical cases. In both methods, candidate execution provides the measurements used to compare performance and deployment cost. The cost of obtaining a skill also matters when its benefits are amortized across later uses. SkillGen (Ma et al., 2026a) and MUSE-Autoskill (Lin et al., 2026) separately report skill construction and subsequent execution costs. These approaches establish that the value of reusable guidance depends on both its acquisition cost and its effect on future execution.

FreeEvo connects efficiency to the evolution decision through two complementary roles. A fixed efficiency atom guides the agent to avoid unnecessary reasoning during execution. The optimizer then uses that guidance to identify redundant behavior in the resulting trajectories and assign negative evidence to implicated accuracy atoms. Changes in output length direct attention to possible inefficiency, while analysis of the trajectory determines which instruction contributed to it. For example, an instruction may help produce a correct answer yet repeatedly trigger a check after the relevant uncertainty has been resolved. Such a trajectory can supply negative evidence for that instruction despite its successful outcome. Efficiency therefore influences both the current reasoning process and the composition of the next skill, supporting adaptation without a separate round of candidate execution and scoring.

B.3 Fine-Grained Experience Management

Experience representations determine how precisely feedback can change future guidance. AutoManual (Chen et al., 2024a) maintains structured rules learned through interaction and compiles them into an instruction manual. Agent Workflow Memory (Wang et al., 2024) induces reusable workflows from past trajectories, while ReasoningBank (Ouyang et al., 2026) distills generalizable strategies from successful and failed experiences and retrieves them for later tasks. These methods organize execution experience into reusable units that can guide behavior across task instances.

ACE (Zhang et al., 2026) makes the update unit explicit by representing its playbook as individual bullets with identifiers and helpful or harmful counters. Reflection produces localized updates, and incremental curation and deduplication maintain the accumulated context. SkillCAT (Chen et al., 2026) further organizes evolved skills into a core and capability nodes with dependency information, enabling task-specific routing and assembly. SkillCommit (He & Yang, 2026) focuses on the validity of abstraction across examples. It uses source and cross-instance replay to assess the compatibility of procedural patches, then retains an expanded skill only when replay supports its claimed scope. These approaches address complementary questions about which experience to retrieve, which entry to revise, and when a procedure can be applied more broadly.

FreeEvo uses modularity to turn efficiency diagnoses into persistent decisions about individual instructions. Each accuracy atom specifies a procedure and its application conditions, giving the optimizer a defined unit to assess. Once evidence is attached, its content is frozen so that later judgments refer to the same instruction. Support accumulates only across distinct examples, repeated visits do not multiply that support, and observed harm replaces earlier positive evidence from the same example. The resulting evidence governs activation and permanent retirement, while the efficiency atom remains fixed. This links the management of reusable experience directly to the reasoning behavior it induces, allowing harmful instructions to be withdrawn while preserving the remaining guidance.

C Experimental Details

Table 5: **Dataset splits and evolution schedules.** The MATH source uses the official training partition. AIME25, AIME26, and MATH-500 are evaluation-only targets. n/a denotes an unused split or schedule.

Dataset	Train	Validation	Test	Evolution steps
Olympiad	135	67	473	4×4
GPQA-D	40	20	138	4×1
TriviaQA Wikipedia	400	200	1400	4×10
ALFWorld	39	18	134	4×1
MATH source	300	150	n/a	4×8
AIME25	n/a	n/a	30	n/a
AIME26	n/a	n/a	30	n/a
MATH-500	n/a	n/a	500	n/a

C.1 Data and Execution Settings

C.1.1 Benchmarks

Tab. 5 lists the fixed partitions and evolution schedules. The first four benchmarks cover mathematical reasoning, scientific question answering, reading comprehension, and interactive task completion. MATH supplies an additional evolution source, while AIME25, AIME26, and MATH-500 serve as evaluation-only transfer targets.

OlympiadBench (Olympiad). OlympiadBench (He et al., 2024) contains olympiad-level mathematics and physics problems. We use a fixed set of 675 mathematical problems presented as text, requiring the agent to derive a final answer through multiple reasoning steps. Predictions are evaluated with the benchmark's automatic answer-equivalence judge, and we report accuracy.

GPQA-Diamond (GPQA-D). GPQA (Rein et al., 2023) contains graduate-level questions written by domain experts in biology, chemistry, and physics. We use its 198-question Diamond subset. Each question presents four answer choices and requires scientific knowledge and reasoning to distinguish the correct answer from distractors. Answer-choice order is fixed across methods, and we report multiple-choice accuracy.

TriviaQA Wikipedia. TriviaQA (Joshi et al., 2017) pairs questions written by trivia enthusiasts with independently collected evidence documents. We use the Wikipedia reading-comprehension setting, supplying the question and its full source evidence to the agent. The task requires identifying the supported answer within this evidence. We report exact match after the official answer normalization, accepting any annotated answer alias.

ALFWorld. ALFWorld (Shridhar et al., 2020) provides household tasks in interactive environments that align textual observations and actions with embodied scenarios. We use its TextWorld environment, where the agent follows a natural-language goal by issuing commands to find, move, and manipulate objects, including cleaning, heating, or cooling them. Each episode requires planning over successive observations and actions. We report success rate using the simulator's task-completion signal.

MATH. MATH contains competition mathematics problems across subjects such as algebra, geometry, number theory, and counting and probability, with worked solutions and final answers. We draw evolution and validation examples from its official training partition. The agent receives the problem statement, and its answer is evaluated with the same answer-equivalence judge used for Olympiad. The resulting skills are transferred to AIME25, AIME26, MATH-500, and Olympiad.

AIME25. AIME25 contains the 30 problems from the 2025 American Invitational Mathematics Examination I and II. These competition problems require mathematical deduction and have integer answers between 0 and 999. We use AIME25 only to evaluate frozen skills transferred from Olympiad or MATH, reporting accuracy by exact integer matching of the final boxed answer.

AIME26. AIME26 contains the 30 problems from the 2026 American Invitational Mathematics Examination I and II. As with AIME25, we evaluate frozen Olympiad and MATH skills without target-side evolution, using exact integer matching of the final boxed answer.

MATH-500. MATH-500 is a 500-problem subset of the original MATH test partition, covering varied mathematical subjects and difficulty levels. We use it as an evaluation-only target for skills evolved on either Olympiad or the MATH training partition, as reported in Tab. 4.

C.1.2 Execution Settings

One epoch covers the source training split, with batches of up to 40 examples. Repeated evaluations use the same test examples; the evaluation counts and exceptions are specified in Appendix C.3.

The Qwen3.6-35B-A3B (Yang et al., 2025) execution configuration enables thinking with temperature 1.0, top-p 0.95, top-k 20, and presence penalty 1.5. The completion limit is 65,536 tokens

for Olympiad, GPQA-D, MATH, AIME25, and AIME26, and 16,384 for TriviaQA and ALFWorld. The execution settings for DeepSeek V4 Flash (Xu et al., 2026) and GLM 4.7 Flash (Zeng et al., 2026) are [XX]. The final optimizer configurations for each method and benchmark are [XX].

C.2 Baseline Settings

SkillOpt. SkillOpt (Yang et al., 2026) retains its trajectory reflection, bounded edits, epoch-level updates, validation gate, and best-skill selection. The w/o Gate variant retains the update pipeline but removes validation rollouts, force-accepts valid updates, and deploys the terminal skill after the prescribed epochs. This also removes selection-based early stopping.

GRASP. For GRASP (Moll et al., 2026), we use the comparison implementation that shares FreeEvo's benchmark adapters and fixed splits. Its library holds at most ten learned skills and supplies at most three per request. The standard method uses four candidate generations and probes of up to twenty examples, followed by validation-based selection among epoch checkpoints. The initial empty-library validation score is recorded separately from this checkpoint competition.

Our w/o Gate variant keeps the four generations, applies structurally valid proposals subject to deduplication and capacity checks, and freezes the last epoch's library. It skips candidate probes, never loads the validation split, and disables validation checkpoint selection and dev-collapse recovery. The latter would otherwise introduce additional probes. Both variants retain the same training schedule and optimizer evidence format.

C.3 Token Accounting and Repeated Evaluation

For a frozen skill s evaluated R times, the reported test output cost is

$$C^{\mathrm{out}}_{\mathrm{test}}(s) = \frac{1}{R}\sum_{r=1}^{R}\sum_{x\in\mathcal{D}_{\mathrm{test}}} L(\tau_x^{s,r}), \tag{11}$$

where r indexes evaluations and L includes every assistant completion in an episode. This is a mean of complete-test totals, reported in millions of tokens. Performance is averaged over the same evaluations. We use $R = 4$ unless noted otherwise in Tab. 1. For ALFWorld, L includes all turns in an episode, including the final success/failure signal.

Evolution cost sums input and completion tokens across task execution, optimizer processing, candidate probes, and method-internal checkpoint selection.

For each setting, performance changes are the method score minus the relevant reference score in percentage points. Relative token changes are $100(C/C_{\mathrm{ref}} - 1)$, using the values reported in the corresponding table. Main and transfer tables use No Skill for performance and Test, and full SkillOpt for evolution cost; ablation changes use full FreeEvo. The cross-model table uses the target model's No Skill and the source model's SkillOpt as references. Its native FreeEvo row reports evolution cost incurred on the target model. Reported averages weight each setting's performance difference or token ratio equally.

C.4 Transfer Protocols

For cross-model transfer (Tab. 3), we apply the frozen Qwen3.6-35B-A3B skills to DeepSeek V4 Flash on Olympiad, GPQA-D, and TriviaQA without further evolution or editing. Native DeepSeek FreeEvo results are reused from Tab. 1.

For cross-dataset transfer (Tab. 4), both source and target executions use Qwen3.6-35B-A3B. Olympiad skills transfer to AIME25, AIME26, and MATH-500; MATH skills transfer to these three targets and Olympiad. Each source skill is frozen and reused across its targets. The Olympiad-to-Olympiad entries are in-domain references, so the transfer averages in Sec. 4.6 cover three Olympiad-source and four MATH-source transfers.

C.5 Fixed Efficiency Guidance

For the experiment in Fig. 3(b), GPT-5.5 generates a short efficiency paragraph for each task. The paragraph is added to the initial skill and remains fixed throughout ungated evolution. We reproduce the exact instructions below. The Olympiad instruction targets repeated reasoning after sufficient support is available. The TriviaQA instruction also governs answer wording, which is directly relevant to its exact-match metric.

Olympiad.

> **Reasoning Efficiency**
>
> - Avoid repeating reasoning that adds no new information. Once the answer is sufficiently supported, give the final answer.

TriviaQA.

> **Reasoning Efficiency**
>
> - Once you know which answer the evidence supports, stop debating descriptive wording: omit words that add no identifying information, while keeping those needed to avoid ambiguity.

D FREEEVO PROMPT TEMPLATES

This appendix gives the prompt templates for the modules in Alg. 1. Each card separates system instructions from the user message; braces denote values filled at runtime. The optimizer receives execution feedback and atom metadata, while the agent receives the composed skill and task input.

Module	When it is used	Prompt
Efficiency atom initialization	After the first execution batch	P1
Accuracy atom discovery	First epoch, with later refinement of atoms without evidence	P2
Task-outcome attribution	After each execution batch	P3
Efficiency attribution	After each execution batch	P4
Skill composition	Before the next execution batch	P5
Task execution	Every agent call	P6–P11

D.1 EFFICIENCY ATOM INITIALIZATION

This template generates a^{eff} from the first batch, executed with $s_0 = \emptyset$ (Appendix A.2). The resulting instruction and conditions remain fixed in subsequent steps.

P1 Efficiency atom initialization

SYSTEM

```
You are FreeEvo's efficiency-instruction designer. Use the first batch
    of task trajectories and feedback to write one reusable efficiency
    atom for this task.

Identify concrete reasoning behaviors that repeat work without resolving
    a new uncertainty. Write an actionable instruction that tells the
    agent when to continue, reuse an established result, or stop.
    Preserve checks and exploration needed to solve the task correctly.

Specify the task type, relevance cues, required assumptions, and
    excluded situations. Ground the instruction in the supplied
    trajectories. Do not include example-specific answers or facts. This
    atom will remain fixed during subsequent evolution.

Return exactly one JSON object with instruction and scope. The scope
    must contain task_type, triggers, preconditions, and exclusions; the
    last three fields are arrays of strings. Return no Markdown or
    additional prose.
```

USER

```
Task description: {task_description}
First-batch feedback: {first_batch_feedback}
```

D.2 ACCURACY ATOM DISCOVERY

New atoms are generated only during the first epoch. Later calls can refine existing atoms without evidence but cannot add new atoms. The input combines current and stored feedback with the atom registry. Each registry entry records whether it has received evidence, allowing the optimizer to preserve its instruction and conditions once they are frozen. Supporting examples motivate a proposal; the evidence-attribution step assesses them before they enter the support sets.

P2 Accuracy atom discovery

SYSTEM

```
You are FreeEvo's accuracy-atom designer. Discover reusable procedures
    from recurring patterns across distinct examples in the supplied
    feedback.

Each atom must describe one concrete procedure and its application
    conditions: task_type, triggers, preconditions, and exclusions. A
    procedure may be demonstrated by successful trajectories or
    motivated by a recurring failure that it would correct. Cite the
    distinct sample IDs and explain the shared pattern.

Compare each proposal with the existing registry. Propose a distinct
    procedure or scope rather than duplicating an existing atom. Never
    include example-specific answers, factual lookup tables, or sample
    IDs in the instruction or scope.

Generate new atoms only during the first epoch, using accumulated
    feedback from its first batch through its last. After that epoch,
    return no new proposals and refine only existing atoms with no
    evidence. Never modify an atom that has received evidence, replace a
    retired atom with a duplicate, or edit the fixed efficiency atom.

Return one JSON object with arrays proposals and refinements. Each entry
    has atom_id, instruction, scope, supporting_sample_ids, and
    diagnostic. A proposal uses a new atom_id; a refinement names an
    existing atom with no evidence. The scope has task_type and string
    arrays triggers, preconditions, and exclusions. Use empty arrays
    when no supported change is available.

New atoms enter as candidates with empty support sets. Do not assign
    evidence counts, confidence scores, activation, or retirement.
    Return no Markdown or additional prose.
```

USER

```
Task description: {task_description}
Evolution phase: {first_epoch_or_later}
Fixed efficiency atom: {efficiency_atom}
Existing atom registry: {atom_registry_with_evidence_status}
Stored feedback: {feedback_history}
Current batch feedback: {batch_feedback}
```

D.3 TASK-OUTCOME EVIDENCE ATTRIBUTION

The optimizer assesses accuracy atoms against the current batch. The input identifies the atoms actually included in the executed skill, so negative attribution respects both activation and the length budget. Positive support can also be assigned to a candidate whose procedure is demonstrated or identified as a relevant correction.

P3 Task-outcome evidence

SYSTEM

```
You are FreeEvo's task-outcome evidence analyst. Assess the accuracy
    atoms against the supplied trajectories and task feedback. Return
    only clear, instruction-specific evidence; omit ambiguous cases.

Use positive polarity with relation observed_success when a successful
    trajectory demonstrates the atom's procedure. Use positive polarity
    with relation corrective when a failed trajectory reveals that
    procedure as a relevant correction. Positive evidence may support a
    candidate that was not included in the executed skill.

Use negative polarity with relation harmful_compliance only when the
    atom was included in the executed skill, was followed or relied on,
    and contributed to task failure. An incorrect answer alone is
    insufficient. Identify the exact instruction and the harmful
    behavior in the diagnostic.

For each nonzero judgment, explain how the trajectory supports the
    assignment. Use only the supplied atom IDs and sample IDs. Do not
    modify instructions or application conditions.

Return one JSON object with an array evidence. Each entry has sample_id,
    atom_id, polarity, relation, and diagnostic. Set polarity to
    "positive" or "negative". Emit at most one entry per sample and
    atom; if both polarities are supported, retain the negative one.
    Omission denotes zero evidence. Use an empty evidence array if no
    clear evidence is available. Return no Markdown or additional prose.
```

USER

```
Accuracy atom registry: {accuracy_atoms}
Atoms included in the executed skill: {selected_atom_ids}
Executed skill: {executed_skill}
Current trajectories and task outcomes: {batch_feedback}
```

D.4 Efficiency Evidence Attribution

This template implements the judgment described in Appendix A.3. For repeated examples, the input includes the previous execution and output-length change. The optimizer attributes concrete redundant behavior to selected accuracy atoms, using the fixed efficiency atom as its reference.

P4 Efficiency evidence

SYSTEM

```
You are FreeEvo's efficiency-evidence analyst. Examine the supplied
    trajectories using the fixed efficiency atom as the reference for
    useful versus redundant reasoning.

For a repeated example, compare its current execution with the previous
    execution and inspect the output-length change. Treat an increase as
    a cue for inspection, not as sufficient evidence of redundancy. If
    there is no previous execution, assess the current trajectory
    against the efficiency atom.

Look for concrete behavior such as repeating a resolved check without
    new information, retrying an unchanged failed approach, or
    repeatedly switching between conflicting instructions. Distinguish
    this from necessary verification, productive exploration, and
    additional work that resolves a genuine uncertainty.

Assign negative efficiency evidence only to an accuracy atom included in
    the executed skill whose instruction can be linked to the redundant
    behavior. Identify the exact instruction and quote the relevant
    trajectory evidence. Pay particular attention to newly activated
    atoms included in this execution. The task may still have succeeded.

Return one JSON object with an array efficiency_evidence. Each entry has
    sample_id, atom_id, polarity, instruction_quote, trajectory_quote,
    and diagnostic. Set polarity to "negative". Emit at most one entry
    per sample and atom. Omit unsupported or ambiguous assignments;
    omission denotes zero evidence. Use an empty efficiency_evidence
    array when no negative evidence is supported.

Do not edit atoms, assign positive efficiency evidence, or change atom
    states. Return no Markdown or additional prose.
```

USER

```
Fixed efficiency atom: {efficiency_atom}
Accuracy atom registry: {accuracy_atoms}
Atoms included in the executed skill: {selected_atom_ids}
Newly activated accuracy atoms: {newly_activated_atom_ids}
Executed skill: {executed_skill}
Current trajectories and task outcomes: {batch_feedback}
Previous executions, where available: {previous_feedback}
Output lengths and available changes: {length_comparisons}
```

The two judgments are combined with negative precedence using Eq. (4). Distinct-example support, scores, and states are then updated by the rules in Appendix A.4. These deterministic updates require no additional prompt.

D.5 Skill Composition and Task Execution

Composed skill. The fixed efficiency atom comes first, followed by the accuracy atoms selected under Appendix A.5. The numbered block below is repeated for each atom. Triggers, preconditions, and exclusions are joined with semicolons; empty fields are omitted. Composition formats the selected instructions and conditions without an additional model call.

P5 Composed skill

```
## Learned Skills

### Skill {index}
Do: {instruction}
Use when: task_type={task_type}; {triggers}; requires: {preconditions}
Do not use when: {exclusions}
```

Mathematical reasoning. OlympiadBench, MATH, AIME25, and AIME26 share the following template. The learned-skill block is omitted when the skill is empty.

P6 Mathematics

SYSTEM

```
You are solving a difficult mathematical olympiad problem.

## Learned skill
{compiled_skill}

Reason carefully and develop a rigorous step-by-step solution. The
learned skill is guidance, not a source of facts; verify every claim
    against the
problem.

Your final answer must appear in `\boxed{...}`. Put every required
    component of
a multi-part answer inside the same final box. Do not box intermediate
    results.
```

USER

```
## Problem
{problem}
```

Scientific reasoning and reading comprehension. GPQA-D receives the question and its labeled choices; TriviaQA receives the source evidence and question. Their skill blocks are omitted when empty.

P7 GPQA-D

SYSTEM

```
You are an expert scientific reasoning agent solving a multiple-choice
    question.

## Skill
{compiled_skill}

## Task Format
You will receive one graduate-level question and four answer choices.
Reason from the information in the question and compare the choices
    carefully.

## Answer Format
Think step by step, then provide your final answer inside
    <answer>...</answer> tags.
Inside the tags, output only one choice label, such as A or C.

Example:
<answer>B</answer>
```

USER

```
## Question
{question}

## Choices
{labeled_choices}
```

P8 TriviaQA Wikipedia

SYSTEM

```
You answer TriviaQA questions using only the supplied source evidence.
    Resolve distractors and aliases carefully. Return exactly one
    non-empty final answer enclosed in <answer>...</answer>; do not emit
    another answer tag, and stop after the tagged answer.

## Frozen skill
{compiled_skill}
```

USER

```
## Source evidence
{context}

## Question
{question}
```

Interactive execution in ALFWorld. At each turn, the nonempty skill prefix precedes the appropriate user-message body. The first turn uses the current observation and admissible actions; subsequent turns additionally provide the task description and recent observation–action pairs. The environment supplies the admissible actions.

P9 ALFWorld: system and skill prefix

SYSTEM

```
You are an expert agent operating in the ALFRED Embodied Environment.
```

USER PREFIX (OMITTED WHEN THE SKILL IS EMPTY)

```
## Skill Knowledge
Below is a skill document with learned strategies. Use these guidelines
    to inform your decisions:

{compiled_skill}
```

P10 ALFWorld: first turn

USER BODY

```
You are an expert agent operating in the ALFRED Embodied Environment.
Your current observation is: {current_observation}
Your admissible actions of the current situation are:
    [{admissible_actions}].

Now it's your turn to take an action.
You should first reason step-by-step about the current situation. This
    reasoning process MUST be enclosed within <think> </think> tags.
Once you've finished your reasoning, you should choose an admissible
    action for current step and present it within <action> </action>
    tags.
```

P11 ALFWorld: later turns

USER BODY

```
You are an expert agent operating in the ALFRED Embodied Environment.
    Your task is to: {task_description}
Prior to this step, you have already taken {step_count} step(s). Below
    are the most recent {history_length} observations and the
    corresponding actions you took: {action_history}
You are now at step {current_step} and your current observation is:
    {current_observation}
Your admissible actions of the current situation are:
    [{admissible_actions}].

Now it's your turn to take an action.
You should first reason step-by-step about the current situation. This
    reasoning process MUST be enclosed within <think> </think> tags.
Once you've finished your reasoning, you should choose an admissible
    action for current step and present it within <action> </action>
    tags.
```