SkillOpt-Lite: Better and Faster Agent Self-evolution with One Line of Vibe

Yifei Shen **Bo Li**[1,2] **Xinjie Zhang**[3]

[1]LMMs-Lab [2]NTU MMLab [3]Microsoft

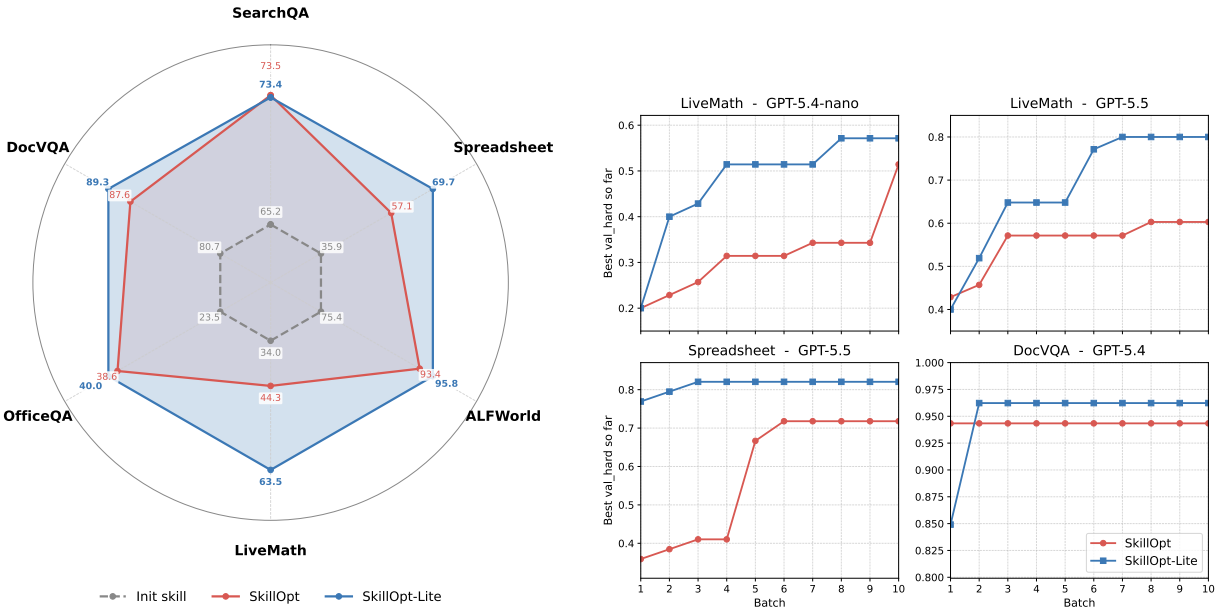
While skill optimization for autonomous agents has gained significant traction, existing methods largely rely on increasingly complex pipelines. This leaves a fundamental question unaddressed: What constitutes a minimal viable pipeline for skill optimization, where every component is justified by theory or empirical necessity? In this note, we first formalize skill optimization through the lens of Zeroth-Order (ZO) optimization, identifying classical counterparts in recent literature such as central difference, trust regions, and variance reduction. However, we identify a key divergence: classical ZO relies on blind numerical perturbations, whereas agentic optimization functions as language-mediated program compilation with trajectories serving as interpretable debugging feedback. Grounded in Claude Code design philosophy and Probably Approximately Correct (PAC) learning, we identify three foundational principles governing the convergence and generalization performance: (1) treating the trajectories as files and exploring them with file system tools; (2) mining consensus attributes across trajectory samples; and (3) enforcing strict independent validation gating. Guided by these principles, we remove the architectural redundancies of existing frameworks to propose **SkillOpt-Lite**, retaining three core components: trajectory exploration, consensus mining, and validation gating. This simplification accelerates convergence and outperforms the complex SkillOpt baseline: on GPT-5.5, it improves the LiveMath performance by **+8.8 points**; on GPT-5.4-nano, it yields a **+25.4 point** increase, allowing the nano model to surpass a standard GPT-5.4 running the full SkillOpt pipeline (55.7 vs. 54.0). Finally, we integrate our framework into production coding agents (e.g., **VSCode Copilot**), enabling developers to evolve agent skills via one line of vibe. Notably, because our framework treats all agent components simply as standard editable code, this minimal pipeline naturally generalizes to full harness optimization—a framework we designate as **HarnessOpt**. On SpreadsheetBench, HarnessOpt allows the lightweight GPT5.4-nano to achieve an accuracy of 0.7758, outperforming the much larger GPT5.5 model running under a standard harness and full SkillOpt pipeline (0.7620).

Code: https://github.com/EvolvingLMMs-Lab/SkillOpt-Lite
Correspondence: yshenaw@connect.ust.hk

1 Introduction

AI agents have transitioned from research prototypes into standard production utilities [1–3]. Practical deployment reveals that an agent's real-world capability is determined not solely by its base Large Language Model (LLM), but by the interaction among the base model, its operational scaffolding (harness), and its domain heuristics (skills). Because foundation models remain frozen during inference, agent engineering frequently reduces to refining these skill documents—a process where subtle textual variations routinely drive non-linear disparities in downstream task performance [4–8].

To manage this non-linear sensitivity, automated skill optimization has evolved from open-loop



(a) Macro-level average performance profiling.

(b) Micro-level validation convergence trajectories.

Figure 1 The overall performance profile and optimization velocity of SkillOpt-Lite. (a) The radar plot illustrates the macro-level capability comparison averaged across all evaluated model scales, showing comprehensive performance gains over both the initial skills and the original SkillOpt. (b) The micro-level convergence curves on representative tracking segments demonstrate that our minimal viable pipeline delivers significantly accelerated policy refinement and higher validation ceilings (`Best val_hard so far`) within a tight 10-batch horizon.

self-reflections [9, 10] into complex algorithmic pipelines [8, 11–13]. Among existing literature, *SkillOpt* [4] formalizes text revision through continuous optimization analogies, implementing mini-batch tree merging, textual learning-rate schedules, and rejected-edit buffers. Yet, despite their empirical gains, existing frameworks exhibit a growing tendency toward architectural complexity. They incorporate intricate mechanics from classical optimization without addressing a fundamental question:

> What constitutes a minimal viable pipeline for skill optimization, where every component is justified by theoretical or empirical necessity?

In this note, we address this question by bridging zeroth-order optimization, statistical learning theory, and systems engineering. We first map existing reflection paradigms to Zeroth-Order (ZO) optimization. As detailed in Table 1, single-trace critiques [9, 10] correspond to 1-point gradient estimators $\hat{\nabla} f(s)$; contrastive diagnoses [13] reflect central difference approximations; fault-isolated edits [11] mirror coordinate descent along basis vectors e_i; while edit budgets [4, 12] and rejected buffers instantiate trust region radii $\mathcal{B}$ and control variates c. However, we identify a key conceptual divergence: classical ZO relies on blind numerical perturbations over unobservable states, whereas agentic rollouts yield explicit, semantic-rich execution traces. Skill optimization functions fundamentally as language-mediated program compilation where trajectories serve as interpretable debugging feedback.

To formalize this compiler paradigm, we establish three foundational design principles. The first two emerge from PAC-learning theory:

(1) **Consensus Mining across Trajectories:** Overfitting to single-trial anomalies inflates the stability coefficient β_{exp}, causing generalization collapse. Instead of executing complex tree-reductions, the optimizer must act as a compression operator capturing cross-task invariants.

(2) **Independent Validation Gating:** PAC bounds demonstrate that a held-out validation set removes β_{exp} from the generalization error. To prevent evaluation bias, gating must rely on independent samples rather than sub-sampled training failures.

Guided by the Unix and Claude Code [14] philosophy of "everything is a file", we design a pilot experiment that isolates each execution trajectory into a standalone flat file and deploys an autonomous coding agent to refine them using primitive file-system tools; empirically, we discover that this single-batch file modification may outperform the full SkillOpt pipeline optimized over four full epochs across multiple benchmarks, leading us to derive our third empirical principle:

(3) **A Bitter Lesson of Skill Optimization:** As base LLMs scale, bespoke topologies designed to damp update velocity become counterproductive. Granting agents primitive shell utilities to directly inspect raw log files consistently outperforms heavily engineered baselines.

Synthesizing these principles, we propose **SkillOpt-Lite**, a minimal viable pipeline that removes mini-batch reflection pooling, slow update damping, and rejection buffers. Instead, *SkillOpt-Lite* operates directly on the disk via an autonomous debugging loop consisting solely of trajectory exploration, consensus mining, and validation gating. As shown in Figure 1, this streamlined design delivers consistent improvements over the fully configured *SkillOpt* baseline across all six benchmarks. On logic-intensive tasks such as Spreadsheet, it yields an average score increase of **+12.6 points** (69.7 vs. 57.1). Furthermore, as tracked in Figure 1b, removing reflection pooling allows lightweight models to circumvent early exploration plateaus; for instance, on ALFWorld, *SkillOpt-Lite* lifts GPT-5.4-nano to **81.3%** (+9.5 over SkillOpt), while demonstrating rapid convergence within the first step on flagship bases (e.g., Spreadsheet-GPT-5.5).

Finally, we encapsulate our pipeline into an open-source VS Code Copilot extension, allowing developers to trigger skill optimization via a single slash command[cite: 1]. Crucially, because our framework treats all agent components simply as standard editable code, lifting file-path restrictions within the IDE environment allows this minimal loop to naturally generalize to full harness optimization—a framework we designate as **HarnessOpt**. On `SpreadsheetBench`, **HarnessOpt** consistently resolves model-specific bottlenecks like reasoning loops. Notably, this framework allows the lightweight `GPT5.4-nano` to achieve an accuracy of 0.7758, outperforming the much larger `GPT5.5` model running under a standard harness and full SkillOpt pipeline (0.7620). In summary, our main contributions are threefold:

- **Theoretical Formalization:** We map agentic skill training to zeroth-order optimization, identifying its unique program-compiler nature, and derive core principles governing stability and validation.
- **Minimal Viable Pipeline:** Guided by the file-centric philosophy, we propose **SkillOpt-Lite**, demonstrating empirically that replacing complex optimization modules with primitive file exploration yields faster convergence and superior performance across multiple baselines.
- **Harness Extension & IDE Encapsulation:** We show that this pipeline directly extends to automated harness optimization (**HarnessOpt**), enabling lightweight models to surpass frontier-class models through environment co-design. We encapsulate the entire workflow into a VSCode extension for one-line agent evolution.

2 Analysis of Skill Optimization

2.1 Skill Training and Zeroth-Order Optimization

2.1.1 Algorithmic Connections

We formalize skill optimization as the maximization of an expected reward function $f(s) = \mathbb{E}_{z\sim\mathcal{D}}[R(H(M, z, s))]$, where $s \in \mathcal{S}_{\text{text}}$ represents a text-based skill artifact, M is a frozen backbone

LLM, z denotes a task instance drawn from distribution $\mathcal{D}$, and H signifies the execution harness. Since the gradient $\nabla_s f$ is analytically intractable due to the discrete nature of $\mathcal{S}_{\text{text}}$ and the non-differentiable composition of H and M, the agent-environment interaction is modeled as a Zeroth-Order (ZO) Oracle. This formulation reveals that existing heuristics utilize components analogous to the classical ZO toolbox [15], despite the semantic domain's lack of a continuous, smooth manifold.

Formally, we establish a structural analogy between continuous ZO operations and discrete text space edits: a text artifact s is evaluated at perturbed positions $s + \mu u$ (or along a standard coordinate basis e_i), where $\mu > 0$ denotes a step size analogue, and u is a random search direction. Under this unified framework, environmental feedback (e.g., execution traces, error logs, and validation scores) acts as the scalar oracle evaluation $f(s)$. LLM-driven self-reflection, patch generation, and skill editing serve as discrete-space structural analogues to continuous ZO operators, such as stochastic gradient estimators $\hat{\nabla} f(s)$, coordinate descent, trust-region constraints $\mathcal{B}$, and variance-reducing control variates c. Consequently, prominent agentic workflows—ranging from single-trace heuristics [9, 10] to structured multi-point variance-reduction schemes [4, 8, 12]—can be systematically analyzed through the lens of algorithmic zeroth-order optimization, as mapped in Table 1.

Table 1 Mapping Between Zeroth-Order Optimization Operators and LLM/Agent Reflection Paradigms

ZO Concept	Formula / Operator	Agent Domain Metric	Literature Realization
Zeroth-Order Oracle	$f(s + \mu u)$	Sandbox Environment Feedback	Scalar metric of training performance.
1-Point Estimate	$\hat{\nabla} f(s) \propto f(s + \mu u)u$	Single-Trajectory Reflection	• **Reflexion** [9]: Direct reflection derived from a single trace. • **Voyager** [10]: Single-error signals triggering local program overwrites.
Multi-Point / Mini-batch	$\frac{1}{b}\sum_{i=1}^{b}[f(s + \mu u_i) - f(s)]u_i$	Batch Rollout Consensus Extraction	• **Trace2Skill** [8]: ZO-SGD using Map-Reduce for patch merging. • **SkillOpt** [4]: Iterative reflection mini-batching with size $B_m = 8$. • **SkillForge** [12]: Phase 2 batch ticket aggregation for trajectory denoising.
Central Difference	$\frac{f(s+\mu u) - f(s-\mu u)}{2\mu}$	Success-Failure Contrastive Analysis	• **SkillCat** [13]: Custom CCE operator for trace contrastive analysis at action divergence point w_i.
ZO Coordinate Descent	$\frac{f(s+\mu e_i) - f(s)}{\mu} e_i$	Fault-Isolated Atomic Modification	• **SkillAdapter** [11]: Coordinate descent fixing faulty step t^* as axis and candidate skill s_j as basis vector.
Trust Region Radius	$\mathcal{B}(s_k, \Delta_k)$	Structural Edit Constraints	• **SkillOpt** [4]: Edit Budget decay ($L_t = 4 \to 2$). • **SkillForge** [12]: Enforcement of the Minimal Modification principle. • **SoftSkill** [16]: Bounding soft-prefix length at $m = 32$ tokens.
Control Variate	$\hat{g}_t - c_t + \mathbb{E}[c]$	Historical Memory Rejection	• **SkillOpt** [4]: Passing rejected edits into a buffer as a negative control variate.

2.1.2 Conceptual Divergence

Although mapping agentic skill training to ZO optimization introduces established variance-reduction methods, a key conceptual divergence exists between the two paradigms. In classical ZO optimization, the decision variable s is defined within a numerical space. The oracle functions as a strict black box that returns only a scalar reward $f(s)$, while the detailed runtime trajectory or intermediate state transitions remain latent and unobservable.

Conversely, the distinguishing characteristic of skill optimization is that every LLM rollout yields an explicit, structured, and semantically rich trajectory containing historical trial logs, intermediate planning rationales, and runtime error messages. These trajectories encapsulate the exact causal chains of success or failure. From this perspective, skill optimization shifts from black-box function querying into a stochastic coding problem, where the skill library serves as a repository of high-level software programs written in natural language. The LLM acts not merely as an evaluator, but as the underlying compiler and execution runtime, while the generated rollout trajectories function as the intermediate program execution traces.

Traditional ZO optimization is forced to approximate gradients blindly through numerical variations because it cannot inspect the underlying function. In contrast, skill optimization leverages readable execution traces to perform targeted, semantic-driven debugging. In Section 3.2, we introduce SkillOpt-Lite, a framework explicitly engineered to exploit this program-compiler paradigm by converting semantic execution traces into actionable code patches.

> **Insight 1 (Conceptual Divergence):** Classical ZO optimization relies on blind numerical perturbations, whereas agentic skill optimization functions as language-mediated program compilation where rollout trajectories serve as interpretable debugging feedback.

2.2 Generalization Error: A PAC-Learning Perspective

Despite its formulation as a language-mediated coding process, skill optimization remains a statistical learning problem governed by the foundational principles of PAC-learning. Under the PAC framework, the generalization error $\epsilon(\mathcal{S})$ of a learned skill library $\mathcal{S}$ over the true task distribution $\mathcal{D}$ is bounded by its empirical error $\hat{\epsilon}_D(\mathcal{S})$ plus a compliance penalty. Instead of relying on uniform capacity measures of the static hypothesis space, this generalization gap can be tightly bounded via the lens of Expected On-Average Stability [17]:

$$\epsilon(\mathcal{S}) \leq \hat{\epsilon}_D(\mathcal{S}) + \mathcal{O}\left(\beta_{\text{exp}} + \sqrt{\frac{\ln(1/\delta)}{N}}\right)$$

where N is the number of training instances, and β_{exp} denotes the expected stability coefficient of the skill refinement algorithm.

Let $D = \{z_1, \ldots, z_N\}$ be the training dataset, and let $D^{\setminus i}$ be the adjacent dataset formed by removing the i-th task instance z_i. An optimization algorithm $\mathcal{A}$ exhibits Expected On-Average Stability with a coefficient β_{exp} if the expectation of its performance variance over the true data distribution $\mathcal{D}$ satisfies:

$$\mathbb{E}_{D,z\sim\mathcal{D}}\left|R(H(M,z,\mathcal{A}(D))) - R(H(M,z,\mathcal{A}(D^{\setminus i})))\right| \leq \beta_{\text{exp}}$$

In the semantic domain, β_{exp} measures the algorithm's statistical resistance to episodic idiosyncrasies. If the refinement process overfits to a specific sample anomaly—such as hardcoding case-by-case branching actions or mimicking local environment variables unique to a single failed trial—the stability coefficient β_{exp} increases, leading to generalization collapse. To minimize β_{exp}, the optimization algorithm must discard single-sample eccentricities and extract the common

attributes across heterogeneous rollouts. This stabilizing compression forces the evolving skills to capture invariant structural logic rather than memorizing single-trial trajectories.

Practical frameworks enforce cross-task consensus through various designs: Trace2Skill [8] and SkillForge [12] aggregate rollouts into mini-batches or batch ticket pools to extract invariant lessons, whereas SkillOpt [4] executes a hierarchical tree reduction via parallelized LLM merging.

Insight 2 (Stability and Overfitting): From the lens of algorithmic stability, a robust skill optimization algorithm acts as a β_{exp}-stabilizing operator, ensuring that text-based skill mutations remain invariant to single-trial rollout anomalies to capture cross-task structural invariants.

To select the optimal model during self-evolution, validation-based model selection is employed. Assuming an evaluation metric bounded by M, the generalization error of the empirically selected skill library complies with a standard model selection bound over m validation samples [18]:

$$\epsilon(\mathcal{S}_{\text{val}}) \leq \hat{\epsilon}_{\text{val}}(\mathcal{S}_{\text{val}}) + \mathcal{O}\left(\sqrt{\frac{\ln(1/\delta)}{m}}\right)$$

By contrasting these two generalization boundaries, we uncover a clear statistical dividend unlocked by the validation protocol. In the standard training bound, the generalization gap is anchored to the algorithm's single-sample sensitivity β_{exp}. As the agent searches through an expressively dense natural language space, the tendency to fit individual sample features penalizes the generalization capacity. Crucially, once a disjoint validation set is introduced, the stability coefficient β_{exp} is completely removed from the model selection upper bound. However, the statistical validity of this decoupling hinges upon a fundamental rule: the validation set must be strictly disjoint and independent from the training set to yield an unbiased estimate of the generalization gap.

An evaluation of recent literature reveals a pervasive violation of this validation protocol: works like Reflexion [9] bypass dynamic validation entirely in an open loop, while frameworks like SkillCat [13], SkillAdapter [11], and Trace2Skill [8] compromise the validation bound by executing their gates either on direct clones of the source training failure instances or on sub-sampled training subsets.

Insight 3 (Independent Validation Set): To guarantee generalization, a validation set must satisfy a dual mandate: strict statistical independence from the training data to prevent evaluation bias, and a sufficient sample size m to suppress the variance bound $\mathcal{O}(\sqrt{\ln(1/\delta)/m})$.

3 SkillOpt-Lite

3.1 A Motivating Example: Writing Skills with Coding Agents

As discussed in Section 2.1.2, skill optimization can be modeled as a program-compiler paradigm, where the skill document serves as the software program and the LLM acts as the underlying compiler. To evaluate this paradigm, we conducted a pilot study mimicking workflows where AI engineers leverage commercial coding assistants to refine agentic skills. Specifically, we collected the raw rollout trajectories generated by the initial batch of a GPT-5.4-nano optimization loop and stored them as local text files. We then deployed GitHub Copilot [19] to autonomously explore this directory solely via primitive file system tools, with the explicit directive to diagnose systematic failure patterns across trajectories and apply minimal modification patches directly to

the baseline skill files.

Crucially, the assistant bypassed all pre-defined mathematical topologies—such as the mini-batch slicing or hierarchical tree-reduction operators utilized in SkillOpt—and committed these edits directly without executing any intermediate validation loops. The refined skills were subsequently evaluated on the downstream test sets of Spreadsheet [20], ALFWorld [21], LiveMath [22], and DocVQA [23].

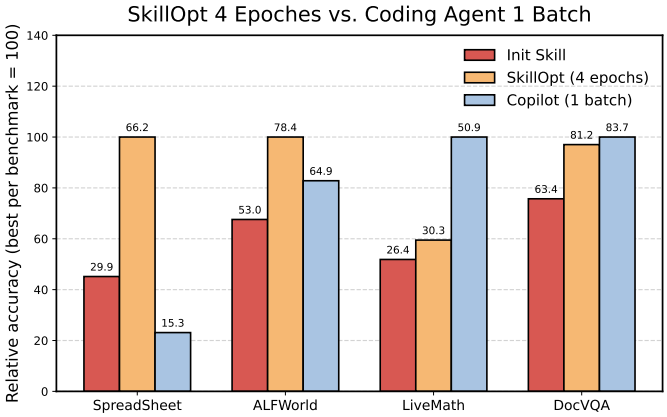


Figure 2 Empirical evaluation of single-batch coding agent exploration versus multi-epoch skill optimization.

The empirical results, illustrated in Figure 2, reveal an instructive phenomenon. Without iterative loops, the coding assistant's single-batch file-system exploration achieved substantial performance gains. Notably, on both LiveMath and DocVQA, this single-batch unvalidated agent configuration outperformed the baseline SkillOpt framework even after the latter underwent four full epochs of iterative optimization. This trend highlights the capability of language models when operating directly over raw execution files. Conversely, on the Spreadsheet benchmark, the optimized skill degraded below its initial baseline performance, demonstrating the necessity of the closed-loop validation mechanisms embedded within the formal optimization pipeline.

Insight 4 (A Bitter Lesson in Skill Optimization): As base models scale, complex algorithmic pipelines designed for skill updates can be consistently matched or outperformed by treating system artifacts as flat files and granting the model primitive file-system navigation tools.

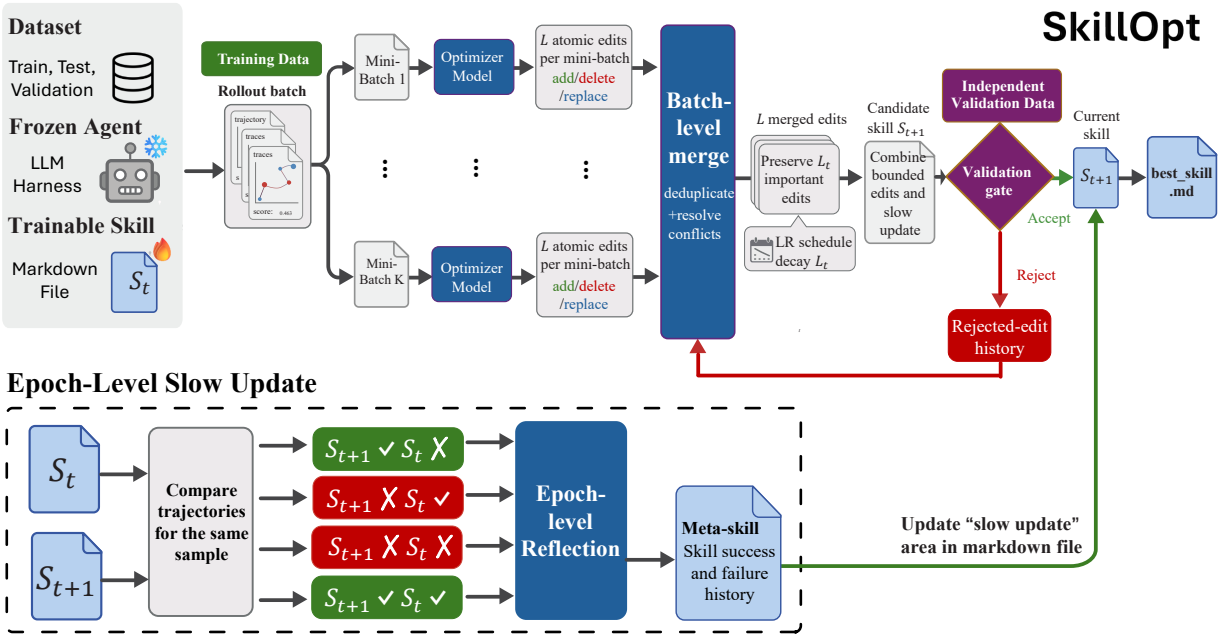
3.2 SkillOpt-Lite Pipeline

We propose **SkillOpt-Lite**, a streamlined framework designed to realize a minimal viable pipeline for skill optimization. As illustrated in the structural comparison in Figure 3, we remove complex batch-level merging, text-based learning rate scheduling, and historical rejected-edit buffers. Furthermore, we deprecate the epoch-level slow update mechanism, which previously required matching and comparing trajectories across multiple epochs for meta-skill reflection.

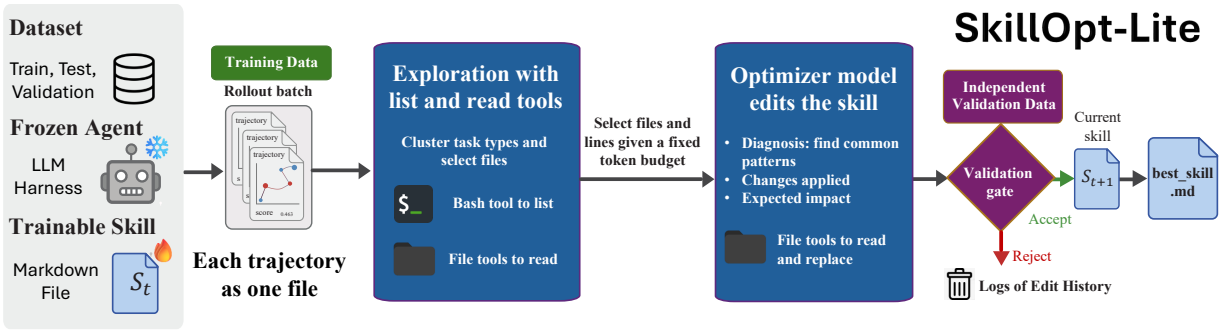
Instead, SkillOpt-Lite operates directly on the local disk via an autonomous debugging loop. The rollout batch is decoupled such that each individual trajectory is stored as a standalone file, enabling the system to navigate, read, and refine the skill library using native file-system tools. The workflow of SkillOpt-Lite is structured as follows:

1. **Trajectory Staging:** Following each batch rollout of the frozen agent harness, the raw trajectory—encompassing planning rationales, environment states, and task scores—is dumped directly onto the disk as a text file. As shown in Figure 3b, each trajectory is stored as an independent log file rather than being aggregated into structural mini-batches.

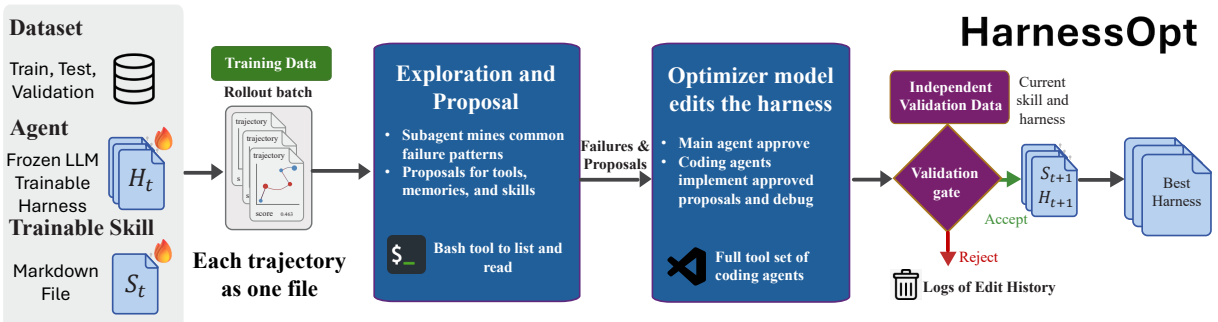
2. **Trajectory Exploration:** Rather than loading the entire corpus of raw logs into the model



(a) The original SkillOpt pipeline [4].



(b) The proposed SkillOpt-Lite pipeline. The framework treats rollout trajectories as independent text files, allowing the optimizer model to explore the directory and extract shared failure patterns. Compared to the baseline SkillOpt framework, our streamlined design eliminates mini-batch reflection pooling, slow update damping, and historical rejection buffers.



(c) The proposed HarnessOpt pipeline. Because SkillOpt-Lite abstracts all artifacts as flat files, the operational harness naturally reduces to editable code files. Consequently, by deploying a coding agent as the core optimizer, the minimal skill pipeline natively generalizes into a full-scale harness optimization framework.

Figure 3 Structural comparison of the optimization pipelines.

context window, the system utilizes explicit file-system utilities to navigate the workspace, adhering to the design principle outlined in Insight 4. Operating under a constrained token budget, the optimizer model executes autonomous shell commands to list directories, cluster uniform task failures, and select high-leverage files for detailed inspection.

3. **Consensus Mining and Minimal Edit:** Equipped with file-reading capabilities, the optimizer reads these trajectory files directly from the directory to discover cross-task invariants and shared failure patterns, fulfilling the objective of Insight 2. Once these patterns are identified, and to satisfy the trust-region constraint of bounded text updates, the system follows a Minimal Update Principle by generating a compact code diff or patch to address the diagnosed errors.

4. **Validation Gating:** The proposed patch is applied to construct a candidate skill library $\tilde{\mathcal{S}}$, which is immediately evaluated on an independent validation set to secure an unbiased empirical score, implementing the verification mechanism of Insight 3. If the candidate skill improves upon the current baseline score, it is accepted as the active skill $\mathcal{S}_{t+1}$. If it additionally exceeds the historical best score, it permanently overwrites the production skill file on disk, designated as `best_skill.md`. Rejected updates are shunted directly to historical logs for archiving.

To demonstrate practical utility, we encapsulate this pipeline into a production-ready VS Code extension. Developers can trigger the full optimization loop directly within their development environment via a single-line slash command:

```
/skillopt-loop rounds=10 batchsize=40 target=gpt5.4-nano [custom_requirements]
```

This integration eliminates the configuration overhead typical of multi-agent platforms, delivering an interactive, lightweight debugging utility directly embedded inside the native IDE environment.

4 Experiments

4.1 Performance

Our experimental configuration follows the evaluation protocols established in SkillOpt, utilizing SearchQA [24], Spreadsheet [20], ALFWorld [21], LiveMath [22], DocVQA [23], and OfficeQA [25]. We employ identical optimizer models for both SkillOpt and SkillOpt-Lite, with GitHub Copilot deployed as the baseline coding agent. Under the original dataset splits, LiveMath and OfficeQA contain only 10 to 20 validation instances, resulting in high validation variance. To mitigate this empirical instability, we adjust the train-validation-test split for these two benchmarks from $2:1:7$ to $2:2:6$. For OfficeQA, we implement an offline evaluation variant due to the absence of active search tool APIs.

Significant Improvements on Reasoning Tasks (LiveMath & Spreadsheet) – On tasks requiring complex reasoning and deterministic code execution, SkillOpt-Lite demonstrates substantial performance gains. For instance, on LiveMath, absolute accuracy increases from 31.2 to 58.8 (+27.6 points) for GPT-4o, and from 36.6 to 73.6 (+37.0 points) for GPT-5.5 relative to the baseline. On Spreadsheet, the streamlined pipeline similarly outperforms the full SkillOpt framework by a clear margin (e.g., 79.4 vs. 61.5 points on GPT-5.4).

Mechanistically, this divergence occurs because the baseline framework relies on mini-batch reflection pooling, an operation that averages multiple distinct textual updates and consequently blurs the implicit gradient signal within discrete language spaces. By eliminating this averaging artifact, SkillOpt-Lite empowers the optimizer model to issue localized, discrete file-system edits that directly resolve logical and algorithmic deadlocks within the skill codebase.

Table 2 Main experimental results across varying model scales. Subscripts denote the relative performance alignment compared with the initial baseline (`Init skill`). Bold text highlights the best performance within each model block. Light blue rows highlight the performance of our proposed `SkillOpt-Lite`. SkillOpt is executed for max(4 epochs, 10 batches), while SkillOpt-Lite is evaluated strictly over 10 batches.

Model	Skill source	SearchQA	Spreadsheet	ALFWorld	LiveMath	OfficeQA (Offline)	DocVQA
GPT-4o	Init skill	64.6	44.1	82.3	25.9	21.0	78.3
	SkillOpt	70.1+5.5	48.7+4.6	95.3+13.0	31.2+5.3	30.2+9.2	85.2+6.9
	SkillOpt-Lite	**71.5**+6.9	**49.8**+5.7	**97.8**+15.5	**58.8**+32.9	**32.4**+11.4	**86.4**+8.1
GPT-5.4-nano	Init skill	50.9	29.9	34.3	26.4	10.8	63.4
	SkillOpt	**68.8**+17.9	51.6+21.7	71.8+37.5	30.3+3.9	**18.4**+7.6	80.4+17.0
	SkillOpt-Lite	66.9+16.0	**66.2**+36.3	**81.3**+47.0	**55.7**+29.3	15.5+4.7	**82.1**+18.7
GPT-5.4-mini	Init skill	69.6	28.2	82.3	34.9	23.0	85.3
	SkillOpt	72.3+2.7	47.5+19.3	**100.0**+17.7	41.2+6.3	**32.1**+9.1	90.9+5.6
	SkillOpt-Lite	**73.1**+3.5	**73.3**+45.1	**100.0**+17.7	**63.2**+28.3	30.4+7.4	**92.5**+7.2
GPT-5.4	Init skill	68.3	39.9	88.1	46.2	29.7	87.7
	SkillOpt	**77.5**+9.2	61.5+21.6	**100.0**+11.9	54.0+7.8	40.2+10.5	90.3+2.6
	SkillOpt-Lite	76.3+8.0	**79.4**+39.5	**100.0**+11.9	**66.0**+19.8	**45.3**+15.6	**91.2**+3.5
GPT-5.5	Init skill	72.4	37.4	90.1	36.6	33.2	89.0
	SkillOpt	78.8+6.4	76.2+38.8	**100.0**+9.9	64.8+28.2	72.2+39.0	91.2+2.2
	SkillOpt-Lite	**79.0**+6.6	**79.7**+42.3	**100.0**+9.9	**73.6**+37.0	**76.2**+43.0	**94.2**+5.2

Consistent Bounds on Semantics-Heavy Domains (SearchQA, ALFWorld & OfficeQA) – Conversely, for tasks dominated by open-ended text retrieval or physical environment grounding, the performance differentials between the two frameworks are marginal. Across SearchQA, ALFWorld, and OfficeQA, SkillOpt-Lite either matches or slightly exceeds the baseline framework, typically fluctuating within a +0.1 to +1.5 point window. This behavior stems from the nature of these benchmarks, which require broad semantic coverage rather than deep, multi-step algorithmic execution.

Because the underlying models rapidly saturate the available optimization margin within these shallow text-action domains, both variants converge toward statistically similar local optima. Crucially, however, our streamlined version achieves this upper bound with significantly reduced computational overhead, demonstrating that complex optimization infrastructure remains largely redundant for shallow, semantic-heavy domains.

4.2 Convergence Speed

To evaluate the optimization efficiency of the proposed minimal pipeline, we analyze the convergence trajectories of SkillOpt and SkillOpt-Lite. Figure 4 tracks the historical best validation scores (Best val_{hard} so far) across 10 sequential optimization intervals under representative model scales and benchmarks.

The empirical trajectories highlight two distinct operational advantages of our streamlined framework:

- **Accelerated Optimization Velocity:** Compared to the full SkillOpt framework, SkillOpt-Lite demonstrates a steeper initial optimization trajectory. For instance, in complex domains such as `LiveMath-GPT-5.5` and `LiveMath-GPT-5.4-nano`, SkillOpt-Lite secures substantial performance gains within the initial 2 to 3 steps. In contrast, the baseline SkillOpt framework exhibits sub-optimal trajectories in early phases, induced by the structural overhead of mini-batch partitioning and slow update damping mechanics.

- **Superior Performance:** Removing the multi-agent rejection buffer and reflection pooling does not induce premature convergence or destabilize policy evolution. Instead, SkillOpt-Lite

consistently establishes an equal or elevated performance ceiling by the final optimization step. As observed in `Spreadsheet-GPT-5.4-nano` and `DocVQA-GPT-5.4`, the autonomous file-system exploration loop enables the optimizer model to isolate structural code defects directly, yielding precise refinements that bypass local sub-optimal states.

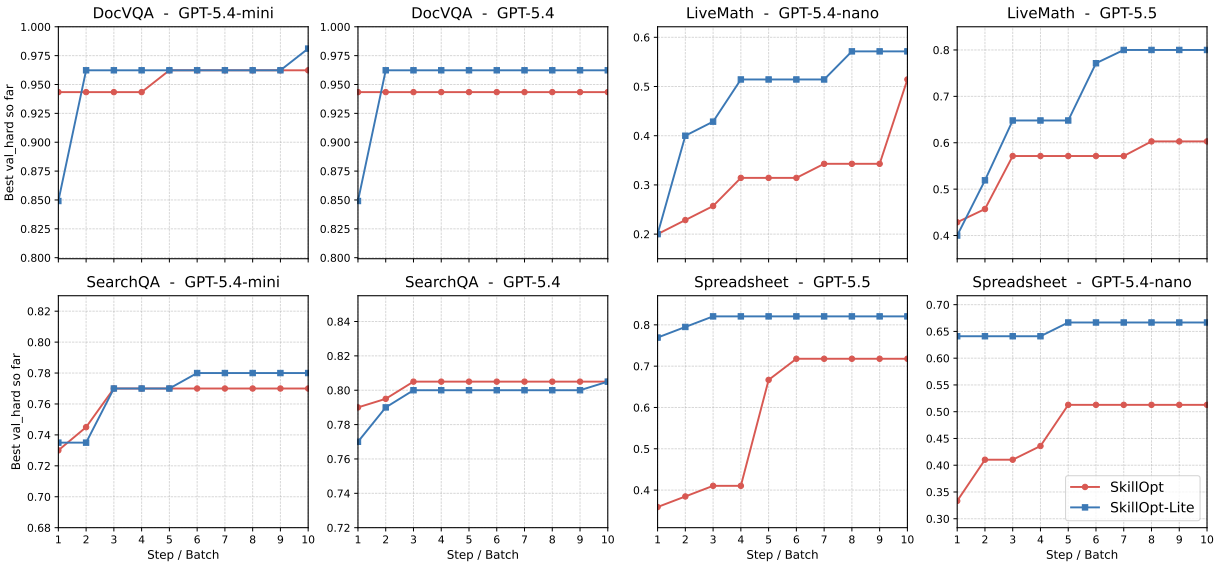


Figure 4 Convergence curve comparison between SkillOpt (red curve with circles) and SkillOpt-Lite (blue curve with squares) across various tasks and model scales over 10 optimization steps. The y-axis denotes the best validation performance achieved so far.

5 Towards Harness Optimization

5.1 HarnessOpt Pipeline

The primary motivation for extending our framework emerges when agentic skill optimization approaches an empirical performance saturation ceiling. Once declarative text prompts and heuristics are optimally tuned, further optimization gains must be unlocked by modifying the imperative control flows and tool execution scaffolding, known as harness optimization [6, 26–28]. Because the core philosophical foundation of *SkillOpt-Lite* treats all agent artifacts simply as standard file-system entities, this minimal pipeline can be natively extended to modify runtime codebase scripts. We designate this fully extended, unified framework as **HarnessOpt**. By lifting file-path restrictions within the workspace directory instead of a single skill markdown file, the optimizer model leverages the identical three-pillar architecture to directly refine the agent's execution routines. An illustration of the pipeline is shown in Fig. 3c.

Round-0 Bootstrapping and Human-in-the-Loop Gate – In harness optimization, the structural integrity of the initial baseline framework—designated as the Round-0 harness—is critical. Blindly mutating raw control logic without a stable initial interface can trigger instant syntax collapse or broken tool dependencies. To address this risk, we implement a structured bootstrapping phase at Step 0. The framework first executes a full rollout of all training instances under the seed harness to aggregate raw system execution traces and exception logs. Next, a specialized diagnostic subagent scans these log directories to perform consensus mining. Its objective is to isolate cross-task architectural deficiencies rather than writing direct code edits, specifically identifying issues such as missing tool primitives or systemic API parsing mismatches. The subagent compiles these diagnostic insights into a structured configuration proposal. This proposal is evaluated through five critical lenses—tool inventory, prompt context, loop policy, codegen-shape, and state memory—culminating in three explicit decisions regarding memory integration, tool

augmentation, and control flow shape. It is rendered as an interactive prompt within the VS Code environment, requiring explicit user approval before any structural alterations are committed to the core codebase.

Automated Continuous Evolution and Sandboxed Gating – Once the Round-0 harness is approved, **HarnessOpt** transitions into a fully automated, closed-loop continuous evolution phase where user intervention is bypassed. In rounds $t \geq 1$, the framework relaxes file permissions, allowing the underlying model to autonomously deploy standard shell tools to inspect execution states and apply targeted code patches to the harness scripts. Crucially, because modifying executable control flows carries the inherent risk of introducing infinite loops or regressions, strict validation and safety guardrails are enforced through three critical loop invariants:

- **Allowlist Constraints**: Modifications are strictly confined to framework scaffolding scripts, while task-specific skills and internal configurations are treated as a read-only denylist to prevent drift.
- **Smoke and Validation Gating**: Validation gating is rigorously performed within an isolated subprocess sandbox. Candidate harnesses must pass a compile check and an early smoke test ($N = 5$) before running the full validation set to measure actual performance gains.
- **Reversible Evolution with Toggles**: All code edits are fully reversible via `git reset` rollbacks, and significant modifications are wrapped behind environment variable feature toggles (initialized at module import time) to ensure clear bisection and future ablation handles.

The validation gate measures the final accuracy score against the historical best configuration. A statistical dead band Δ depending on the sample number is established to filter out stochastic variance. Patches yielding improvements below this deadband are accepted only if they demonstrate non-trivial progression on the continuous secondary soft metrics; otherwise, the state is rolled back to prevent codebase inflation with non-functional artifacts.

To seamlessly integrate this capability into the developer workflow, we expose the harness optimization pipeline as a production-level slash command within our environment. Developers can initiate the continuous evolution loop by providing the targeted iteration rounds, the batch size for train rollouts, the target base model, and a reference skill file. A typical invocation follows the format below:

```
/harnessopt-loop rounds=2 batchsize=40 target=gpt5.4-nano skill=best_skill_nano.md
[custom_requirements]
```

5.2 Experimental Evaluation on Harness Optimization

In our preliminary experiments, we observed that for the majority of the six datasets used in our skill optimization benchmark, the existing infrastructure was already sufficient, or harness optimization yielded minimal changes. Therefore, we focus our evaluation on `SpreadsheetBench` as a representative and challenging case study to analyze the efficacy of **HarnessOpt**. We conduct evaluations across four language models of varying capacities: `GPT5.4-nano`, `GPT5.4-mini`, `GPT5.4`, and `GPT5.5`. The detailed results are shown in Table 3, which corresponds to the empirical data in image_0d4db6.png.

Analysis of Model Behavioral Changes – By applying **HarnessOpt**, we identified distinct operational bottlenecks across different model tiers and fixed them with targeted harness modifications:

- **`GPT5.4-mini` and `GPT5.4`**: For these two models, the baseline performance was largely limited by insufficient observation of the data files and weak final answer verification. **HarnessOpt** addressed this by expanding the visible range of the spreadsheet preview and introducing a

Table 3 Optimization results on `SpreadsheetBench` across different LLM baselines.

Method	GPT5.4-nano	GPT5.4-mini	GPT5.4	GPT5.5
Init skill	0.2989	0.2821	0.3986	0.3737
SkillOpt	0.5160	0.4750	0.6150	0.7620
SkillOpt-Lite	0.6619	0.7331	0.7936	0.7972
HarnessOpt w.o. skill	0.7651	0.8114	0.8363	0.8363
HarnessOpt w. skill	**0.7758**	**0.8256**	**0.8505**	**0.8577**

dedicated step for the agent to introspect its final outputs. This successfully reduced parsing and formatting errors.

- **`GPT5.5` and `GPT5.4-nano`**: In contrast, these two models frequently encountered scenarios where their reasoning chains would get stuck in repetitive loops when a tool execution failed. To break this deadlock, **HarnessOpt** automatically intercepted these repetitive states and applied a fallback policy (`retry reasoning=low`), allowing the models to recover and complete the tasks.

The Importance of Joint Optimization – As shown in Table 3, optimizing the harness codebase alone (*HarnessOpt w.o. skill*) significantly boosts performance over purely prompt-based optimization (*SkillOpt-Lite*), raising `GPT5.4-nano` from 0.6619 to 0.7651 and `GPT5.5` to 0.8363. Notably, with harness optimization, the lightweight `GPT5.4-nano` achieves 0.7758 (*HarnessOpt w. skill*), outperforming the much larger `GPT5.5` model running under a basic harness with full prompt optimization (*SkillOpt*, at 0.7620). This capability inversion demonstrates that refining environmental scaffolding can empower a smaller model to surpass a frontier-class model in a suboptimal environment. Crucially, the best overall results are achieved only when we optimize both components simultaneously (*HarnessOpt w. skill*), reaching 0.8505 for `GPT5.4` and 0.8577 for `GPT5.5`. This indicates that while skills provide strategic guidelines, the harness ensures a reliable execution environment; real-world performance relies on their co-design.

6 Conclusion and Future Work

6.1 Conclusion

In this note, we formalize agentic skill training through the lens of zeroth-order optimization and statistical learning theory. We show that complex multi-agent pooling and text update-damping mechanisms are largely redundant as base models scale. Guided by "everything is file" philosophy, we propose SkillOpt-Lite, a minimal viable pipeline that treats rollout trajectories as independent flat files and leverages autonomous coding agents for targeted, semantic-driven debugging. Empirically, SkillOpt-Lite accelerates optimization convergence and consistently matches or outperforms heavily engineered baselines across multiple benchmarks. Finally, we encapsulate this workflow into a native IDE extension, demonstrating that file-centric skill editing can naturally generalize to full harness optimization HarnessOpt.

6.2 Future Work

To move toward fully autonomous agent self-evolution, we outline four practical directions for future research:

- **Skill Optimization for Frontier Model Distillation:** While distilling proprietary frontier models into open-weight equivalents is a standard industrial practice, active anti-distillation defenses and sub-optimal generation boundaries often hinder data quality. Utilizing skill optimization to refine teacher agents across granular capabilities provides a structured framework for generating

high-quality distillation corpora. However, because this paradigm couples dataset generation with the active optimization of downstream architectures, designing fast, compute-efficient evaluation protocols for data selection remains a key challenge.

- **Harness Template Database for Automated Optimization:** While this work explores basic programmatic modifications of the execution harness, scaling harness optimization requires a robust reference framework. Developing a curated library of minimal harness templates is essential to provide effective structural priors for coding agents. Furthermore, heterogeneous harnesses require adaptive sandboxing to ensure secure interaction. While current state-of-the-art agents (e.g., Claude Code [14]) offer effective safety protocols for local software engineering environments, extending these containment mechanisms to internet-augmented or multimodally grounded execution domains remains unaddressed.

- **Harness Evolution as Continual Learning:** Given the high computational overhead of continuously fine-tuning foundational weights, operating directly on the non-parametric layer—specifically treating execution harnesses and domain skills as the evolving parameters—presents a promising alternative for lifelong learning [29]. A primary challenge under this framework lies in the structural alignment and fusion of independently evolved lineages of harnesses over extended temporal horizons. A direct extension of this research involves developing personalized agents, where the harness architecture continuously adapts to match long-term human preferences and idiomatic developer workflows.

- **Extending Optimization to Foundation Model Training:** While the base foundation models remain frozen in our current framework, this optimization pipeline can naturally extend to the model training phase itself. In our philosophy, since data collections are managed as standard files, a foundation model represents an intrinsic compilation of these files. By using our identical three-pillar architecture, the framework can establish a comprehensive benchmark to stress-test model boundaries, automatically edit web-scraping controllers or data-distillation prompts to gather targeted samples, and use this data to train an iteratively better model. This shifts the optimization boundary from scripts to model parameters, achieving full agent self-evolution. Interestingly, we have already pioneered semi-automated versions of this approach, such as dynamically scaling image-text pairs during the pre-training of our internal multimodal generation and understanding models, and curating specific video training data during our development of LLaVA-OneVision-2 [30] (though these structural implementation details remained unhighlighted in the main paper). Fully automating this data-model-harness co-design loop without manual intervention remains a critical next step.

References

[1] Anthropic. Claude code: An agentic command line tool. https://docs.anthropic.com/en/docs/agents-and-tools/claude-code/overview, 2025. Accessed: 2026-06-24.

[2] OpenClaw Contributors. Openclaw: An open-source agentic command-line assistant. https://github.com/openclaw/openclaw, 2025. GitHub Repository.

[3] Nous Research. Hermes agent. https://nousresearch.com/hermes3/, 2026. Advanced Agentic and Function-Calling Foundation Models.

[4] 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.

[5] 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.

[6] Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

[7] Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

[8] 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.

[9] 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.

[10] 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.

[11] Zhuoyun Yu, Xin Xie, Wuguannan Yao, Chenxi Wang, Lei Liang, Xiang Qi, and Shumin Deng. Skilladaptor: Self-adapting skills for llm agents from trajectories. *arXiv preprint arXiv:2606.01311*, 2026.

[12] Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. Skillforge: Forging domain-specific, self-evolving agent skills in cloud technical support. *arXiv preprint arXiv:2604.08618*, 2026.

[13] 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.

[14] Anthropic. Claude Code Source Code. https://github.com/codeaashu/claude-code, 2026.

[15] Sijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Gaoyuan Zhang, Alfred O Hero III, and Pramod K Varshney. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine*, 37(5):43–54, 2020.

[16] Xijia Tao, Yihua Teng, Xinyu Fu, Ziru Liu, Kecheng Chen, Yuzhi Zhao, Suiyun Zhang, Rui Liu, and Lingpeng Kong. Softskill: Behavioral compression for contextual adaptation. *arXiv preprint arXiv:2606.20333*, 2026.

[17] Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Learnability and stability in the vapnik-chervonenkis sense. *The Annals of Statistics*, 38(5):2642–2696, 2010.

[18] Tong Zhang. *Mathematical Analysis of Machine Learning Algorithms*. Cambridge University Press, 2023. doi: 10.1017/9781009093057.

[19] GitHub. GitHub Copilot. https://github.com/features/copilot, 2021. Accessed: 2026.

[20] 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.

[21] 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.

[22] 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.

[23] Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2200–2209, 2021.

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

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

[26] Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026.

[27] Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026.

[28] Harness Self-Evolution. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents. 2026.

[29] Jiayi Weng. Learning beyond gradients. https://trinkle23897.github.io/learning-beyond-gradients/, May 2026. Accessed: 2026-06-30.

[30] Xiang An, Yin Xie, Feilong Tang, Yunyao Yan, Huajie Tan, Didi Zhu, Changrui Chen, Xiuwei Zhao, Bin Qin, Kaicheng Yang, et al. Llava-onevision-2: Towards next-generation perceptual intelligence. *arXiv preprint arXiv:2605.25979*, 2026.