Hexo Labs

SIA: Self Improving AI with Harness & Weight Updates

Prannay Hebbar*[1], Yogendra Manawat*[1], Samuel Verboomen[2], Alesia Ivanova[4], Selvam Palanimalai[3], Kunal Bhatia[1], Vignesh Baskaran[1]

*Equal contribution. · [1]Hexo Labs Palo Alto · [2]Hexo Labs Brussels · [3]Hexo Labs Toronto · [4]University of Oxford

Keywords: Self-Improving Agents, Test-Time Training, Reinforcement Learning, Harness Engineering, Scaffold Generation

Abstract

Humans are the bottleneck in building and improving AI. Both the models and the agents that wrap them are written, tuned, and corrected by people. The long-horizon goal of an AI that can figure out how to improve itself remains open. Two largely disjoint research lines attack this bottleneck. The *harness-update school* has a meta-agent rewrite the scaffold of a task-specific agent (its tools, prompts, retry logic, and search procedure) while the model weights are held fixed. The *test-time training school* uses hand-written RL pipelines to update the model's own weights on task feedback while the harness is held fixed. These two silos operate in isolation. We propose **SIA**, a self-improving loop in which a language-model agent (the **Feedback-Agent**) updates *both* the harness *and* the weights of a task-specific agent. We evaluate across three contrasting domains: Chinese legal charge classification, low-level GPU kernel optimisation, and single-cell RNA denoising. Combining both levers outperforms scaffold iteration alone on all three benchmarks. SIA-W+H achieves **25.1%** over prior SOTA on LawBench, **12.4%** faster GPU kernels than prior SOTA (1,017 vs 1,161 μs), and **20.4%** over prior SOTA on denoising. Harness updates make the model agentic, shaping how it searches and acts, while weight updates build the domain intuition that no prompt or scaffold can instil.

1. Introduction

1.1. Humans are the bottleneck.

Today's progress in AI is rate-limited by humans. The models are designed and post-trained by researchers, and the agents built on top of them are scaffolded, prompted, debugged, and tuned by engineers. The long-horizon goal of the field an AI (model or agent) that can figure out how to improve itself remains open. We treat this paper as one concrete step toward that goal: a system that, given only a task specification and a verifier (both defined in §3), improves *both* its scaffold and its model weights without further human intervention.

1.2. Two silos of self-improving AI.

Research into automated self-improvement has bifurcated into two largely disjoint silos as follows.

Silo 1 Harness/scaffold self-improvement. A meta-agent rewrites the scaffold of the task-specific agent its system prompt, tool-dispatch logic, retry policy, and answer-extraction code across generations, while the underlying language-model weights are held fixed. Recent representatives include the Darwin

Gödel Machine (Zhang et al., 2025), Meta-Harness (Lee et al., 2026), Hyperagents (Zhang et al., 2026), and the broader line on automated agentic system design (Hu et al., 2024). The recurring empirical observation in this silo is that scaffold edits concentrate on software-engineering hygiene parsing, retries, dispatch and rarely deliver domain-specific reasoning that the base model could not produce given any prompt.

Silo 2 Test-time post-training. A hand-written RL pipeline updates the model's own weights on task feedback at test time, typically with the harness held fixed at a single prompt-and-grader template. Representatives include TTRL (Zuo et al., 2025), the *Discover* line of test-time training (Yuksekgonul et al., 2026), and the surprising-effectiveness-of-TTT result (Akyürek et al., 2024). Here the gain comes from internal policy change, but the pipeline that delivers it is engineered by humans and does not adapt to the task structure that a scaffolded agent would expose.

The gap. These two silos operate in isolation. Harness work leaves the model fixed; test-time training leaves the harness fixed.

1.3. Contributions.

- We propose and evaluate a Feedback-Agent that **also trains** the task-specific agent's weights, in combination with scaffold updates, to improve performance on arbitrary downstream tasks. The system is task-agnostic: given a task specification and a verifier, it produces both an evolved scaffold and an RL-adapted set of Low-Rank Adaptation (LoRA; Hu et al., 2022) weights.
- We empirically demonstrate the combined approach across **three contrasting domains** law (191-class Chinese charge classification; Fei et al., 2023), systems (Triton kernel optimisation on H100; Novikov et al., 2025), and biology (single-cell RNA denoising; van Dijk et al., 2018) and observe **consistent gains over prior SOTA**: 25.1% on LawBench (70.1% vs 45.0%), 12.4% faster GPU kernels (1,017 vs 1,161 μs), and 20.4% on denoising (0.289 vs 0.240 `mse_norm`).
- We isolate the **harness-only** contribution (harness update trajectories across several iterations) and contrast it with the full pipeline (harness + weight updates), demonstrating that weight updates deliver gains beyond what the harness alone achieves.

1.4. Roadmap.

§2 states the research questions the paper answers and maps each to a later section. §3 defines the technical vocabulary. §4 places SIA in the landscape of self-improving and test-time-training work. §5 describes the configurable-loop method. §6 presents the per-task results and ablations. §7 discusses what each lever changes. §8 and §9 close with limitations and future work.

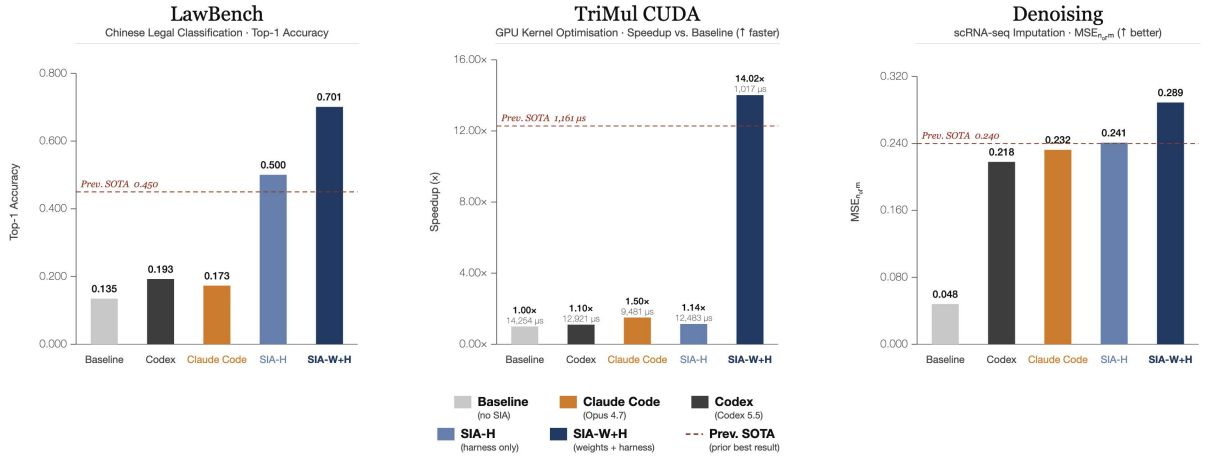

Figure 1. SIA across three diverse tasks. Each panel compares three operating points: Baseline (first generation, no SIA), SIA-H (harness updates only), and SIA-W+H (harness + weight updates), on LawBench Top-1 accuracy, TriMul CUDA speedup, and scRNA-seq denoising `mse_norm`. The dashed line marks the previous state-of-the-art. SIA-W+H strictly outperforms SIA-H on all three tasks.

2. Research Questions

This paper is organised around two research questions. Each is answered by a specific later section.

- **RQ1 Overall thesis.** We first ask how much harness iteration alone improves a task-specific agent when model weights are held fixed. We then ask whether running both levers together (iteratively updating the harness *and* the model weights in a single loop) pushes past that harness-only ceiling. Does the combined approach outperform scaffold iteration alone, and does this hold across contrasting domains?
- **RQ2 Mechanism: what does each lever change?** Do weight updates surface domain knowledge that no scaffold edit reaches, and does harness iteration produce qualitatively different (external infrastructure) changes?

3. Background and Preliminaries

3.1. Agent and its components.

A **task-specific agent** is a program that takes a task instance and produces an answer. We decompose it into:

- **LLM.** The underlying language model with weights θ. We use `openai/gpt-oss-120b` as the base model throughout.
- **System prompt.** Fixed text prepended to every model call that frames the task.
- **Tool-dispatch logic.** Python code that parses model tool-call outputs and routes them to handlers (file I/O, code execution, dataset lookup, grader calls).
- **Answer extraction.** Code that converts a model response (typically a structured trailing block) into a benchmark-formatted prediction.
- **Grader.** The deterministic verifier the orchestrator invokes to compute the per-instance reward.

We call the fixed, non-weight component of the agent the **scaffold** (equivalently, **harness**) throughout. It is the union of the system prompt, tool-dispatch logic, answer extraction, and any supporting infrastructure, every part of the agent that is fixed code rather than model output.

3.2. Meta-agent vs. task-specific agent.

A **meta-agent** is an LLM call whose output is itself an agent. SIA uses two meta-agents:

- **Meta-Agent ($\mathcal{M}$).** Generates the initial scaffold A_1 from the task specification $\mathcal{U}$ and any reference implementations $\mathcal{R}$ supplied with the benchmark:

$$A_1 = \mathcal{M}(\mathcal{U}, \mathcal{R}).$$

- **Feedback-Agent ($\mathcal{F}$).** Reads the previous generation's scaffold A_g, its execution trajectory τ_g, and performance metrics $\mathcal{E}_g$, and synthesises an improved scaffold:

$$A_{g+1} = \mathcal{F}(A_g, \tau_g, \mathcal{E}_g, \mathcal{U}).$$

The **task-specific agent** is the scaffold A_g at generation g that actually executes against the evaluation dataset.

3.3. Trajectory and feedback loop.

Unlike systems that condition improvement on aggregate metrics alone, $\mathcal{F}$ receives the full **trajectory** τ_g, the complete structured execution log from running A_g against $\mathcal{D}$ (the evaluation dataset): every prompt,

model response, tool call, tool result, and extracted answer for every task instance. This allows $\mathcal{F}$ to diagnose specific failure modes rather than react to summary statistics.

Each generation g follows a three-phase protocol:

1. **Execution.** A_g runs on $\mathcal{D}$ inside a sandbox: read-only access to the dataset directory, read/write access to a working directory. The trajectory τ_g is captured.
2. **Analysis.** $\mathcal{F}$ receives A_g's source code, τ_g, the metrics $\mathcal{E}_g$, and optionally sample task descriptions used to discourage single-instance overfitting.
3. **Improvement.** $\mathcal{F}$ emits two artefacts: an *improvement report* (prose analysis and the proposed changes) and the *next-generation agent* A_{g+1}.

3.4. Symbol table.

Symbol	Meaning
g	Generation index
$G_{\max}$	Maximum number of generations
A_g	Agent scaffold at generation g
$\mathcal{D}$	Evaluation dataset
$\mathcal{U}$	Task specification (benchmark description + sample instances)
$\mathcal{E}_g$	Performance metrics and error logs at generation g
τ_g	Execution trajectory at generation g
$\mathcal{F}$	Feedback agent
G	Number of rollouts per state during RL training
π_θ	Current policy (model with trainable weights θ)
π_{θ_0}	Frozen reference policy (base model)
s	Initial state (task prompt)
a	Action (model-generated response / rollout)
$V(s, a)$	Task reward for action a given state s

4. Related Work

We survey each silo, characterise the specific gap SIA addresses, and summarise the landscape in a comparison table.

4.1. Harness / scaffold self-improvement.

- **Darwin Gödel Machine (Zhang et al., 2025).** Evolutionary search over agent source code: a population of agents proposes and evaluates code mutations to themselves, with the highest-fitness variants surviving. The model is fixed.
- **Meta-Harness (Lee et al., 2026).** LLM-driven harness mutation with end-to-end optimisation of the harness graph. SIA's harness update step is closest to Meta-Harness in spirit; the difference is that we follow harness convergence with weight updates rather than further mutation.
- **Hyperagents (Zhang et al., 2026).** The closest concurrent work. Hyperagents allows the *meta-mechanism* itself the rules by which the meta-agent edits the task-specific agent to be editable, not just the task-specific agent. The agent and the agent-improver coevolve. The distinction from SIA is the lever: Hyperagents adds expressivity to scaffold edits but leaves the model weights fixed; SIA adds a second, weight-based lever.
- **AI Scientist (Lu et al., 2024).** A full research-pipeline meta-agent that proposes hypotheses, runs experiments, writes papers. The agent's outputs are research artefacts, not modified scaffolds; the scaffold is held fixed across runs.
- **Automated design of agentic systems (Hu et al., 2024).** Meta-search over compositions of building blocks (sub-agents, tools, prompts). Model fixed.

- **AutoResearcher (Karpathy, 2026).** A static scaffold for autonomous ML experimentation: the agent proposes and runs experiment configurations, but the agent architecture itself does not change across iterations.

4.2. Test-time training and test-time RL.

- **Learning to discover at test time (Yuksekgonul et al., 2026).** The objective we use in training update steps. Trains weights at test time using rollouts under an entropic-utility objective; SIA reuses this loss and the LoRA-based training stack.
- **Surprising effectiveness of TTT (Akyürek et al., 2024).** Empirical demonstration that per-task gradient adaptation at test time substantially improves few-shot performance. Establishes the TTT-as-adaptation framing.
- **TTRL (Zuo et al., 2025).** RL on unlabelled test data using majority-vote-derived pseudo-rewards. The setting is single-prompt, single-response; there is no scaffold and no per-instance verifier. SIA differs in that the reward is a deterministic task verifier and the rollout is scaffolded.
- **STaR (Zelikman et al., 2022); Self-Refine (Madaan et al., 2023); Reflexion (Shinn et al., 2023).** Earlier self-improvement loops that bootstrap reasoning traces or use verbal critique. STaR fine-tunes the model on self-generated rationales (a supervised weight update); Self-Refine and Reflexion operate purely at inference time with no weight updates.
- **Self-play fine-tuning (Chen et al., 2024).** Iterative fine-tuning where the model's own outputs serve as training signal. The training pipeline is hand-written; the scaffold is fixed.
- **EUREKA (Ma et al., 2023).** An LLM generates reward functions (a scaffold-side change), which are then used to train RL policies (a weight-side change). The two components interact, but the reward-function generator is not itself updated by the trained policy, the loop is one-directional rather than co-evolutionary. SIA differs in that the Feedback-Agent dynamically selects between scaffold and weight updates in a closed feedback loop, with each update type informed by trajectories produced under the current state of both components.

4.3. RL and agent training infrastructure.

Across all training runs, we use `gpt-oss-120b` with LoRA rank 32 as the base model and adapter configuration. Weight updates are executed on H100 GPUs via **Modal**, our RL training platform, which handles rollout generation, reward assignment, and gradient updates within a single managed pipeline. SIA builds on existing training frameworks; the Feedback-Agent composes these infrastructure components under its control, treating weight updates as one of two selectable actions alongside scaffold rewriting. Related infrastructure includes **verl/HybridFlow** (Sheng et al., 2024) for flexible RLHF, **SkyRL** (Cao et al., 2025) for long-horizon agent training, **LLaMA-Factory** (Zheng et al., 2024) for unified post-training, and **Axolotl** for streamlined fine-tuning configurations.

4.4. Comparison table.

Table 1. Comparison of self-improving / automated agents along two axes. Does the system edit the harness? Does it edit the model weights?

Agent	Edits harness	Edits weights
SIA (ours)	**Yes**	**Yes**
Hyperagents (Zhang et al., 2026)	Yes	No
Darwin Gödel Machine (Zhang et al., 2025)	Yes	No
Meta-Harness (Lee et al., 2026)	Yes	No
AI Scientist (Lu et al., 2024)	Partial	No
Automated agentic system design (Hu et al., 2024)	Yes	No
AutoResearcher (Karpathy, 2026)	No	No
TTRL (Zuo et al., 2025)	No	Yes
Discover-TTT (Yuksekgonul et al., 2026; Akyürek et al., 2024)	No	Yes
EUREKA (Ma et al., 2023)	Partial	Yes
FunSearch (Romera-Paredes et al., 2024)	Partial	No
Voyager (Wang et al., 2023)	Yes	No
Self-Refine (Madaan et al., 2023) / Reflexion (Shinn et al., 2023)	Partial	No
STaR (Zelikman et al., 2022)	No	Yes
ReAct (Yao et al., 2022)	No	No

SIA is, to our knowledge, the only entry that updates both the scaffold and the weights in a single self-improving loop.

5. Method

5.1. Overview.

SIA is a configurable loop driven by three LLM components: a Meta-Agent, a Task-Specific Agent, and a Feedback-Agent. The Meta-Agent initialises the task-specific agent's scaffold. After each execution, the Feedback-Agent observes the trajectory and performance, then dynamically selects, at each step, between two complementary actions: a **harness update** (scaffold evolution with weights fixed) or a **training algorithm update** (weight update via an RL method of the Feedback-Agent's choosing, with the scaffold fixed). The choice of action, and the choice of training algorithm when a weight update is selected, are conditioned on task type and observed reward dynamics. **Harness Update Phase** and **Weight Update Phase** are soft labels for these two action types, not rigid sequential stages.

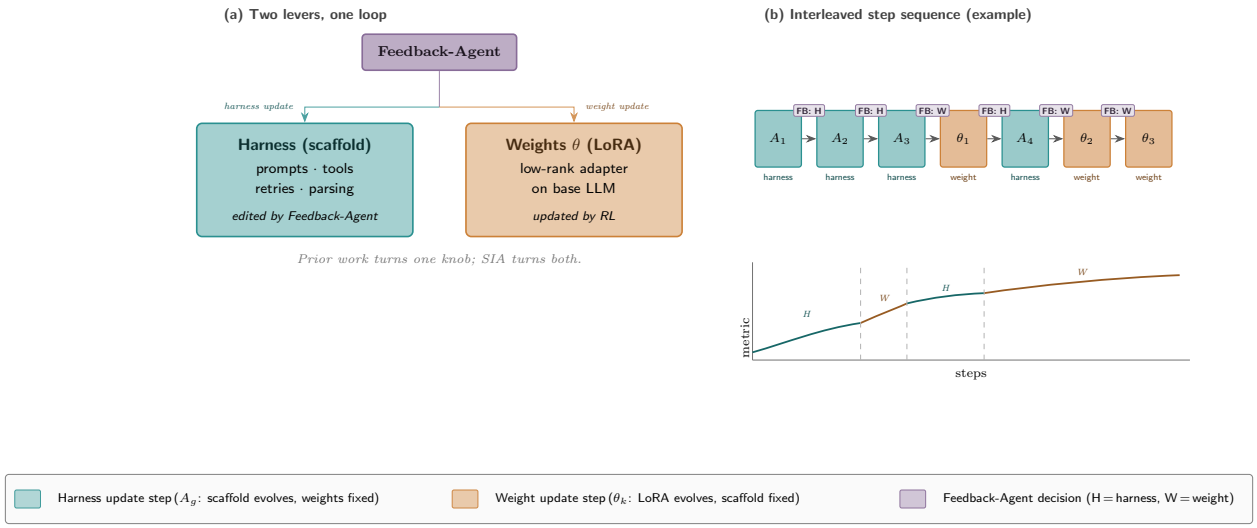

Figure 2. Conceptual view of SIA. (a) Two complementary levers (a textual scaffold and a LoRA adapter). After each execution, the Feedback-Agent (mauve) selects the next action: a harness update (teal) or a weight update (amber). The two levers are interleaved freely, not locked into sequential phases. (b) An example 7-step sequence showing the Feedback-Agent alternating between harness and weight updates. Each **FB:H**/**FB:W** badge marks one decision. The metric curve rises from both types of step, with harness updates (teal segments) and weight updates (amber segments) each contributing distinct gains.

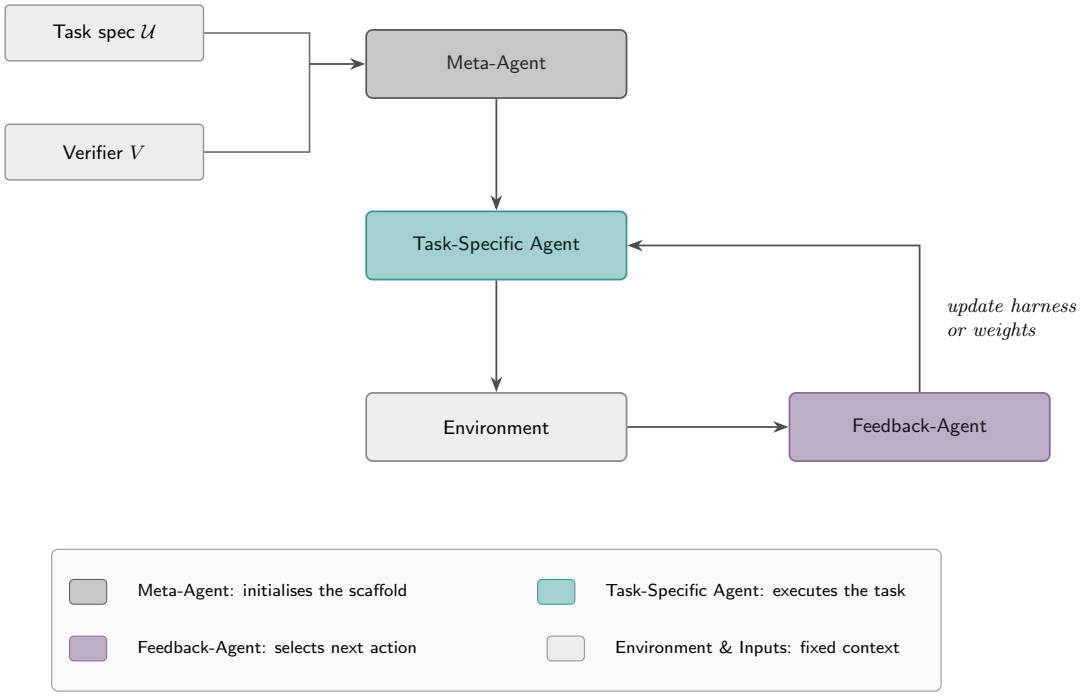

Figure 3. SIA system architecture. The Meta-Agent initialises a scaffold from the task specification $\mathcal{U}$ and verifier V. The Task-Specific Agent executes inside the Environment, producing a trajectory; the Feedback-Agent analyses the trajectory and selects the next action, either synthesising an improved scaffold (harness update) or triggering a weight update, then feeds the result back to the Task-Specific Agent. The loop repeats until the step budget is exhausted.

5.2. System components.

SIA's three components (Meta-Agent, Task-Specific Agent, Feedback-Agent) are defined in §3.2; implementation details follow here. Across all experiments, the Meta-Agent and Feedback-Agent use `Claude Sonnet 4.6`; the task-specific agent uses `gpt-oss-120b` (harness steps) or an RL-adapted checkpoint thereof (training steps).

5.3. Harness updates.

When the Feedback-Agent selects a harness update, the loop runs one scaffold evolution step. Each such step follows the per-step protocol (Execution → Analysis → Improvement). Rollouts are produced by the current model π_θ (base or RL-adapted); the model weights θ are held fixed during this step and only the scaffold A_g changes. The recurrence is

$$A_{g+1} = \mathcal{F}(A_g, \tau_g(\pi_\theta), \mathcal{E}_g, \mathcal{U}),$$

where $\tau_g(\pi_\theta)$ denotes trajectories collected by executing scaffold A_g with model π_θ.

Sample-task regularisation. The Meta-Agent is conditioned on a diverse set of task specifications during scaffold generation, which mitigates overfitting the initial scaffold to a single benchmark instance.

6. Experiments

We evaluate SIA on three contrasting tasks spanning law, systems, and biology. These benchmarks are commonly used to evaluate other self-improving AI systems; we run on them specifically to enable direct comparison against prior work.

6.1. Setup.

Table 2. Per-task evaluation setup.

Task	Domain	Train / Test	Metric	Previous SOTA	Verifier
LawBench (191-class)	Chinese legal	5,332 / 913	top-1 accuracy	0.450	held-out test-split grader
AlphaEvolve TriMul	Low-level	n/a / fixed input shape	score = 1500/runtime (higher = faster)	1.292	H100 timing
MAGIC scRNA-seq Denoising	Single-cell	n/a / pancreas scRNA-seq	`mse_norm` ($\in [0,1]$, higher = better)	0.24	MAGIC reference against ground truth

All weight update steps adapt `gpt-oss-120b` via LoRA (rank $r = 32$, learning rate 4×10^{-5}).

6.2. Baselines.

Because harness update steps start from a meta-agent-initialised scaffold around `gpt-oss-120b` and run against the same verifier we report, **the initial score is, by construction, `gpt-oss-120b` filtered through the Meta-Agent's initial scaffold A_1 (a task-specific system prompt, single-tool dispatch loop, and output parser generated once from the benchmark specification before any Feedback-Agent iteration begins)**. The harness update trajectory then traces what scaffold iteration adds on top of that baseline, and the weight update trajectory traces what weight updates add on top of the harness-only best. We treat this as our primary baseline structure. Across all tasks, the Feedback-Agent

begins with scaffold iteration and switches to weight updates once harness progress stalls; we report **SIA-H** (harness-only best) and **SIA-W+H** (harness + weight updates best) to isolate each lever's contribution.

6.3. Per-task results.

6.3.1. LawBench: 191-Class Chinese Criminal Charge Classification.

LawBench (Fei et al., 2023) is a multi-class legal document classification benchmark drawn from real Chinese criminal case descriptions. Given a factual case summary, the model must identify the correct criminal charge from 191 distinct categories in Chinese statutory law. The 191 classes encode fine-grained legal distinctions that even trained practitioners find demanding: categories of theft (ordinary theft, public-property theft, embezzlement), assault (simple, aggravated, grievous bodily harm), and fraud variants each differ in legally precise factual elements with direct consequences for sentencing. A random-guess baseline is correct less than one percent of the time. The benchmark contains 5,332 training samples and 913 test samples; all evaluations are on the held-out test split.

Harness updates. Early scaffold iterations established a working classification pipeline; subsequent generations restructured it around a TF-IDF + LinearSVC pipeline, iteratively tuning the character n-gram range and regulariser C, steadily improving accuracy until gains levelled off at **50.0%**, a **36.5 percentage point gain** over the initial run. At this point the Feedback-Agent detected stalling reward and switched to weight updates.

Weight updates. Because the reward signal is a clean outcome-based scalar (correct charge or not) and rollouts — each a generated solution script executed against the test split — are cheap to generate in parallel, the Feedback-Agent applied PPO with GAE: a learned value head V_ϕ produces per-token advantage estimates over the generated solution script, and a clipped surrogate objective $\min(r_t\hat{A}_t, \text{clip}(r_t, 1 \pm \varepsilon)\hat{A}_t)$ prevents the policy from drifting far from a working baseline. This applied stable gradient pressure on the model's ability to write classification code that correctly handles fine-grained charge distinctions, pushing accuracy to **70.1%**, an additional **20.1 percentage point gain** over the harness-only best (Figure 4).

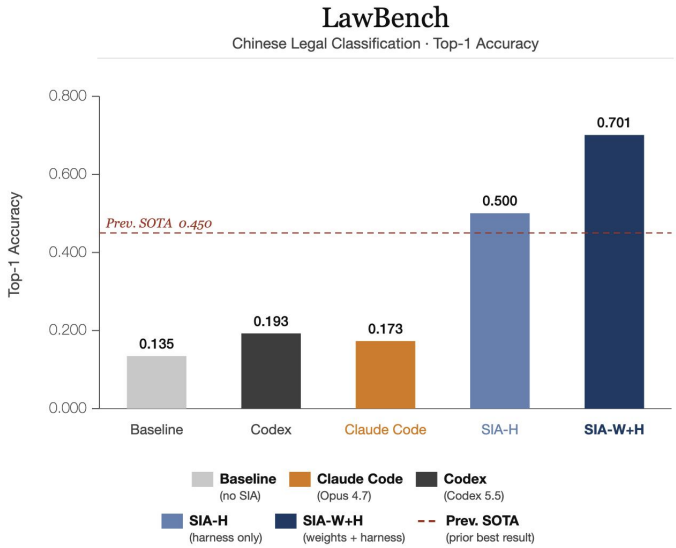

Figure 4. LawBench results. Top-1 accuracy for Baseline, SIA-H (harness only), and SIA-W+H (harness + weight updates). Dashed line: prior state-of-the-art.

6.3.2. AlphaEvolve TriMul: CUDA Kernel Optimisation for Protein Structure Prediction.

The triangular multiplicative update (TriMul) is a core operation in AlphaFold2's Evoformer module, used to propagate pairwise residue-interaction features during protein structure prediction. The task, drawn from the AlphaEvolve benchmark, asks an agent to write a custom CUDA kernel for this operation on an H100 GPU. TriMul is memory-bandwidth-limited rather than compute-limited: threads access non-contiguous

memory due to the triangular sparsity structure, inducing warp divergence and cache misses that defeat standard dense-matrix optimisation techniques. Achieving high throughput requires H100-specific knowledge, tensor core scheduling, shared-memory tiling, register pressure management, that standard libraries (cuBLAS, cuSPARSE) do not apply to this operation. Score is defined as 1500/runtime, so a higher score means a faster kernel.

Harness updates. The agent progressively built and refined working CUDA kernels across iterations, converging on a best runtime of **12,483**, a **1.14**× speedup. Incremental scaffold changes (memory layout hints, compilation flags, retry logic) continued to yield smaller gains until the trajectory plateaued, at which point the Feedback-Agent switched to weight updates.

Weight updates. Kernel optimisation has a sparse, outcome-heavy reward structure: most generated kernels either fail to compile or are far from optimal, making raw gradient signal from a cold start uninformative. The Feedback-Agent applied entropic advantage weighting, which up-weights high-reward rollouts and discounts near-zero-reward noise via softmax redistribution with adaptive temperature β: $w_i \propto \exp(r_i/\beta)$, enabling productive gradient flow even when most kernels in a rollout batch are poor. This allowed the model to internalise H100-specific design patterns, shared-memory tiling, fp32 register accumulation, block-size selection, that no scaffold edit could encode, driving runtime down to **1,017** and a final speedup of **14.02**×, a **91.9% reduction** from the harness-only peak (Figure 5).

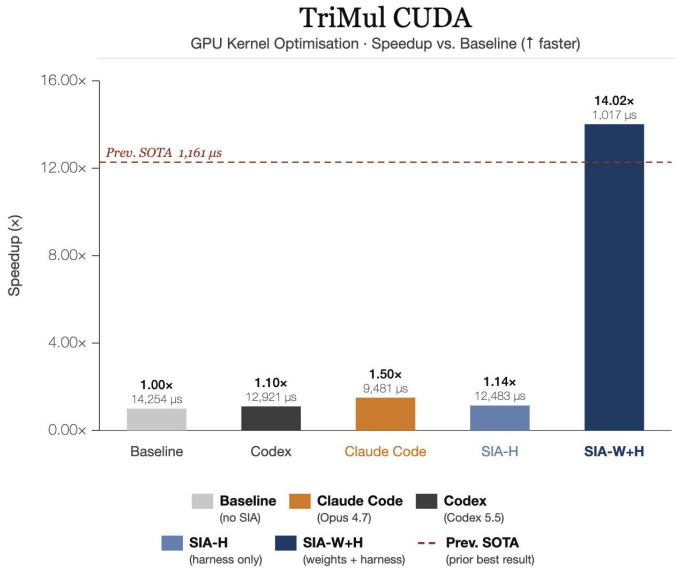

Figure 5. TriMul CUDA results. Speedup over baseline for Baseline, SIA-H (harness only), and SIA-W+H (harness + weight updates). Dashed line: prior state-of-the-art.

6.3.3. MAGIC scRNA-seq Denoising: Single-Cell RNA Imputation.

Single-cell RNA sequencing (scRNA-seq) measures gene expression across thousands of individual cells, but the resulting count matrices are highly sparse: many true non-zero counts are observed as zero due to technical dropout. MAGIC (Markov Affinity-based Graph Imputation of Cells) addresses this by constructing a k-nearest-neighbour graph over cells, computing Markov transition probabilities, and diffusing expression values across graph neighbours to impute missing signal. The task asks an agent to tune MAGIC's coupled hyperparameters, number of neighbours k, diffusion steps t, kernel bandwidth α, and preprocessing choices, on pancreas scRNA-seq data (Baron et al., 2016). The optimisation is non-trivial: k too small overfits to individual cell noise; too large causes over-smoothing that destroys true biological signal. Evaluation uses `mse_norm`, a normalised reconstruction quality score against ground truth (higher is better; 1.0 is perfect imputation).

Harness updates. The agent swept the coupled hyperparameter space of MAGIC, neighbours k, diffusion steps t, bandwidth α, across several iterations and reached a stable plateau, with `mse_norm` settling at a best

of **0.241**. Further scaffold iterations produced no meaningful improvement, prompting the Feedback-Agent to switch to weight updates.

Weight updates. Using GRPO, the model moved beyond parameter tuning entirely. Crucially, the first weight-update checkpoint introduced a structural transformation that the scaffold-only loop, across all harness iterations, never generated: a two-line post-processing step (`np.clip` + `np.rint`) that rounds imputed counts to non-negative integers, enforcing a biological invariant that is trivially correct yet absent from any prior scaffold version. This lifted `mse_norm` to **0.289**, a **20% gain** over the harness-only best (Figure 6).

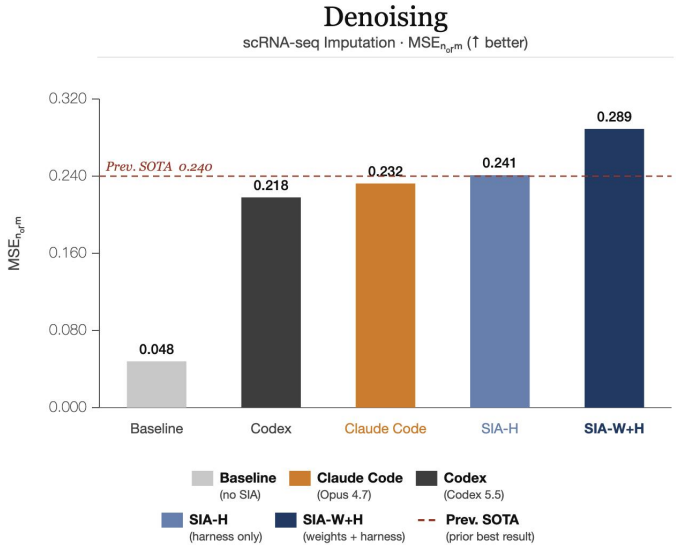

Figure 6. Denoising results. MSE_{norm} for Baseline, SIA-H (harness only), and SIA-W+H (harness + weight updates). Dashed line: prior state-of-the-art.

7. Discussion

7.1. Combined vs. harness-only (RQ1)

To isolate each lever's contribution we ablate SIA-H (harness updates only) against SIA-W+H (harness + weight updates). Table 3 reports the initial score, prior SOTA, and both operating points across all three tasks.

Table 3. Ablation: SIA-H vs. SIA-W+H. "Initial" is the `gpt-oss-120b` score through the Meta-Agent's initial scaffold A_1. SIA-H is the harness-only best; SIA-W+H adds weight updates.

Task	Initial	Prev. SOTA	SIA-H (harness only)	SIA-W+H (harness + weights)
LawBench (top-1 acc)	13.5%	45.0%	50.0%	**70.1%**
AlphaEvolve TriMul (reward)	0.105	1.292	0.120	**1.475**
Denoising (`mse_norm`)	0.048	0.240	0.241	**0.289**

SIA-W+H strictly outperforms SIA-H on every task, confirming RQ1. The gains are substantial: **+20.1 pp** on LawBench, **91.9%** runtime reduction on TriMul (12,483 → 1,017 μs), and **20%** on denoising. Each lever occupies a distinct change space, external scaffold versus internal parameters, so neither saturates the gain available from the other (see §7.2–7.4).

7.2. What does harness iteration change? (RQ2a)

Harness iteration produces *externalised* changes, new tools, tighter parsers, search procedures, retry policies, and prompt structure, while model weights stay fixed.

Across the three tasks, the Feedback-Agent was observed building increasingly specialised scaffolding: on LawBench, a structured answer-extraction layer and an SVC re-ranker over the model's top candidates; on TriMul, a compilation-error parser that fed CUDA diagnostics back as structured context and a timing harness returning median runtime; on MAGIC denoising, a batched configuration driver and a result-parsing tool that organised (parameter-set, score) pairs for the model to reason over.

In all three cases, the changes are software-engineering improvements: new tools, tighter output parsers, smarter retry logic. The model checkpoint is unchanged throughout; all gains come from how the scaffold mediates between the model and the task environment.

7.3. How the Feedback-Agent applies weight updates (RQ2b)

The Feedback-Agent does not run a fixed RL procedure. While this paper reports three tasks, we have been running SIA across a wider set of tasks not included here; the algorithm descriptions below reflect common patterns observed across that broader experimentation. In the three tasks reported, PPO with GAE was observed on LawBench; entropic advantage weighting on TriMul; and GRPO on denoising. A fuller treatment of algorithm selection across tasks is deferred to v2.

- **PPO with GAE.** *Observed when:* step-level rewards are dense and training stability is the binding constraint, multi-step tool-use or long code-generation tasks where a single catastrophic update would collapse the policy. A learned value head V_ϕ produces per-token advantage estimates $\hat{A}_t = \sum_l (\gamma\lambda)^l \delta_{t+l}$; a clipped surrogate $\min(r_t \hat{A}_t, \text{clip}(r_t, 1 \pm \varepsilon)\hat{A}_t)$ prevents the policy from leaving the trust region. The dual actor-critic optimisation is expensive but yields the lowest-variance gradient signal available.
- **GRPO.** *Observed when:* rollouts are cheap to sample and the verifier fires at episode end, classification, short-answer, or unit-test tasks where hundreds of completions can be scored in a single forward pass. Advantages are normalised within a rollout group of size G: $\hat{A}_i = (r_i - \bar{r})/\sigma_r$, eliminating the value network entirely. This halves memory and enables large parallel batches.
- **Entropic advantage weighting.** *Observed when:* the reward histogram is heavily right-skewed, tasks where correct solutions are rare but individually high-signal, such as hard mathematical proofs or low-pass-rate code synthesis. Rather than zeroing out below-average rollouts, gradient mass is redistributed via softmax with adaptive temperature β: $w_i \propto \exp(r_i/\beta)$ (Yuksekgonul et al., 2026). The temperature is tuned online so that the effective sample size stays above a floor threshold, preventing collapse onto a single trajectory.
- **REINFORCE + KL-to-base.** *Observed when:* the reward is dense and the primary risk is capability regression rather than gradient variance, fine-grained domain-adaptation tasks where the base model is already near-capable and large parameter movement is undesirable. Monte Carlo returns $R_t = \sum_{t' \geq t} \gamma^{t'-t} r_{t'}$ serve as advantages directly, augmented with a penalty $\alpha\,\text{KL}(\pi_\theta \| \pi_{\theta_0})$ against the frozen reference. No critic, no grouping, the simplest possible training loop.
- **Best-of-N behavioural cloning.** *Observed when:* reward is so sparse that $\mathbb{E}[r] \approx 0$ across all rollouts and policy gradient signal is numerically zero. The Feedback-Agent invokes this as a phase-zero cold-start: the top-k rollouts by verifier score are distilled into the model via cross-entropy loss, raising the baseline pass rate to a level where a subsequent PPO or GRPO phase becomes viable.
- **DPO.** *Observed when:* the verifier can rank outputs but not score them absolutely, tasks with soft quality criteria where ordinal signal is reliable but cardinal reward is not. Given a winning rollout y^+ and a losing rollout y^-, the objective $-\log\sigma\left(\beta \log \frac{\pi_\theta(y^+)}{\pi_{\theta_0}(y^+)} - \beta \log \frac{\pi_\theta(y^-)}{\pi_{\theta_0}(y^-)}\right)$ is minimised directly without a reward model.

Each training approach is conditioned on trajectory observations rather than a fixed schedule.

7.4. What weight updates change (RQ2b)

Weight updates produce *internalised* knowledge: domain-specific patterns encoded into the model's parameters that no scaffold edit reaches. Unlike harness changes, which modify the infrastructure surrounding the model, weight updates modify the model's prior over solutions directly.

On LawBench, gradient pressure on the 191-class charge taxonomy sharpened the model's disambiguation of adjacent categories, distinguishing theft sub-types, assault grades, and fraud variants, without any prompt-side hint. On AlphaEvolve TriMul, the weights converged on H100-specific kernel design patterns (shared-memory tiling, fp32 register accumulation, block-size selection) that the base model never produced regardless of scaffold quality. On MAGIC denoising, the first weight-update checkpoint introduced a structural invariant the harness had never proposed: a `np.clip` + `np.rint` post-processing step that rounded imputed counts to non-negative integers, encoding a biological constraint directly into the policy.

In each case the internalised knowledge is task-specific and verifier-aligned, it emerges from direct gradient pressure, not from any human-authored instruction. The harness shapes *how* the agent searches; weight updates change *what* the model knows.

8. Limitations

Coupled co-evolutionary Goodhart. Harness search and RL weight updates both optimise against the same fixed verifier V. Each pass shapes the distribution the other sees: the harness finds scaffolds that are easy for the current policy to exploit; the weights train on data collected through a scaffold that will subsequently change. The joint fixed point of this coupled system is a Nash equilibrium between two optimisers that are blind to each other's update history, not a point that maximises V on out-of-distribution scaffolds or novel policies. Standard Goodhart analyses assume a single optimiser; the two-lever setting produces a *coupled* variant whose fixed points can appear strong on the training verifier while being fragile under any perturbation to either component.

9. Future Work

Meta-RL over the action-selection policy. The Feedback-Agent currently selects between harness and weight updates using a frozen LLM prior. A more principled approach treats the selection policy itself as the object to be learned: run SIA across a distribution of tasks, treat each (trajectory, action, outcome) triple as a transition in an outer MDP, and train the selector via RL on that outer MDP. The selector then improves its lever-attribution through experience across tasks rather than relying on fixed heuristics calibrated on trajectories from a different capability regime. This creates a genuinely recursive structure, a self-improving system whose improvement mechanism is itself self-improving, and raises non-trivial questions about the stability of such nested loops that are distinct from any question in single-level RL or meta-learning.

More interleaved training and harness switching. The current SIA loop alternates between harness search and weight update phases in discrete, coarse-grained rounds. A finer-grained schedule, where the Feedback-Agent can trigger a weight update mid-harness search, or resume harness exploration immediately after a gradient step, could reduce the lag between observing a plateau and acting on it, and may unlock improvement trajectories that coarse alternation misses.

References

[1] Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. *International Conference on Machine Learning*, 2024.

[2] Maayan Baron, Adrian Veres, Samuel L. Wolock, Abigail L. Faust, Renaud Gaujoux, Amedeo Vetere, Jennifer Hyoje Ryu, Bridget K. Wagner, Shai S. Shen-Orr, Allon M. Klein, et al. A single-cell transcriptomic map of the human and mouse pancreas reveals inter- and intra-cell population structure. *Cell Systems*, 3(4):346–360.e4, 2016.

[3] Shiyi Cao et al. SkyRL-v0: Train real-world long-horizon agents via reinforcement learning. Technical report, NovaSky, UC Berkeley, 2025.

[4] Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. In *International Conference on Machine Learning*, 2024.

[5] D. V. Dijk, Roshan Sharma, J. Nainys, Kristina M. Yim, Pooja Kathail, Ambrose J. Carr, Cassandra Burdziak, Kevin R. Moon, Christine L. Chaffer, D. Pattabiraman, et al. Recovering gene interactions from single-cell data using data diffusion. *Cell*, 174(3):716–729.e27, 2018.

[6] Zhiwei Fei, Xiaoyu Shen, D. Zhu, Fengzhe Zhou, Zhuo Han, Songyang Zhang, Kai Chen, Zongwen Shen, and Jidong Ge. LawBench: Benchmarking legal knowledge of large language models. *arXiv.org*, 2023.

[7] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

[8] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *International Conference on Learning Representations*, 2024.

[9] Andrej Karpathy. autoresearch: AI agents running research on single-GPU nanochat training automatically, 2026.

[10] Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *International Conference on Machine Learning*, 2026.

[11] Chris Lu, Cong Lu, Robert Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv.org*, 2024.

[12] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2023.

[13] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Neural Information Processing Systems*, volume 36, pages 46534–46594. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023.

[14] Alexander Novikov, Ngân V˜u, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, S. Shirobokov, B. Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv.org*, 2025.

[15] B. Romera-Paredes, M. Barekatain, Alexander Novikov, Matej Balog, M. P. Kumar, Emilien Dupont, Francisco J. R. Ruiz, J. Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2023.

[16] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *European Conference on Computer Systems*, 2024.

[17] Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Neural Information Processing Systems*, volume 36, pages 8634–8652. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023.

[18] 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. *Trans. Mach. Learn. Res.*, 2023.

[19] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

[20] Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, et al. Learning to discover at test time. *arXiv.org*, 2026.

[21] Eric Zelikman, Yuhuai Wu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems 35*, volume 35, pages 15476–15488. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022.

[22] Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. *SuperIntelligence - Robotics - Safety & Alignment*, 2(3), 2025.

[23] Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026.

[24] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, and Yongqiang Ma. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Annual Meeting of the Association for Computational Linguistics*, pages 400–410. Association for Computational Linguistics, 2024.

[25] Yuxin Zuo, Kaiyan Zhang, Shang Qu, Li Sheng, Xuekai Zhu, Yuchen Zhang, Biqing Qi, Youbang Sun, Ganqu Cui, Ning Ding, et al. TTRL: Test-time reinforcement learning. *arXiv.org*, 2025.