

OASIS: Online Anchor-calibrated Self-Improvement System

Wenyi Wang^{1,†}, Firas Laakom¹, Li Nanbo¹, Yanning Dai¹, Yuhui Wang^{1,†},
Yifan Xu¹, Dmitrii Khizbullin¹, Piotr Piękos¹, Yilan Zhang¹, Yimeng Chen¹,
Alfonso Amayuelas⁵, Mingchen Zhuge¹, Jürgen Schmidhuber^{1,2,3,4}

¹ King Abdullah University of Science and Technology (KAUST),

² Dalle Molle Institute for Artificial Intelligence Research (IDSIA), ³ Università della Svizzera italiana (USI),

⁴ Scuola universitaria professionale della Svizzera italiana (SUPSI), ⁵ University of California, Santa Barbara.

[†] Corresponding authors. Email: {yuhui.wang, wenyi.wang}@kaust.edu.sa

Abstract

Many recursively self-improving systems improve by editing their codebases based on observed rollout behavior. However, recent methods largely treat self-improvement as offline search: variants are optimized and evaluated on fixed benchmarks. This misaligns with real-world deployment, where tasks arrive sequentially and may have different distributions. We study recursive self-improvement as online learning over non-stationary task streams. In this setting, reliable execution and fallback are particularly difficult with limited evidence: different branches of self-improvement see different tasks, so raw rewards conflate capability with task difficulty, leading to overly conservative or optimistic reversion. We propose Online Anchor-calibrated Self-Improvement System (OASIS) that runs an adaptive reference agent on incoming tasks, converts raw rewards into paired comparisons, and uses confidence-calibrated rules for branch continuation and fallback without evaluating every agent on all tasks. Across SWE-Bench Pro and DeepResearch Bench, OASIS consistently improves over both the initial agent and prior RSI baseline SSA, raising the SWE-Bench Pro solve rate from 51.6% to 56.9% using Qwen3.6-35B-A3B and DeepResearch Bench from 35.18 to 39.31 using Gemini 3 Flash. This enables OASIS to outperform larger open weights and proprietary systems, including Qwen3.5-397B-A17B, Minimax 2.7, Qwen3.6-Plus, and GPT-5.3-Codex, with sustained deep self-improvements rather than shallow cycles.

1 Introduction

Recent progress in language-model agents (Yao et al., 2022; Schick et al., 2023; Wu et al., 2024) and their optimization (Khatab et al., 2023; Zhuge et al., 2024; Hu et al., 2024) has renewed interest in recursively self-improving artificial intelligence (Schmidhuber, 1987; Schmidhuber et al., 1997): systems that can modify their own prompts,

tools, memory, code, or control logic in order to improve future performance and their capacity to further self-improve (Zhang et al., 2026a; Wang et al., 2026). This line of work revisits a long-standing ambition in AI: to reduce reliance on human-designed agent architectures and enable agents to autonomously discover better mechanisms for acting, reasoning, and improving themselves (Turing, 1950; Good, 1966). Recent approaches such as the Darwin Gödel Machine (Zhang et al., 2026a) and the Huxley-Gödel Machine (Wang et al., 2026) demonstrate the practical promise of this direction by allowing coding agents to modify their own codebases and retain variants that improve benchmark performance.

Despite this progress, recent recursive self-improvement systems are still largely framed as offline agent discovery: variants are generated, evaluated under fixed benchmark protocols, and selected by benchmark performance (Robeyns et al., 2025; Zhang et al., 2026a; Wang et al., 2026; Zhang et al., 2026b). This is natural when the objective is to find a strong final agent, as candidates can be tested on comparable tasks, failures discarded, and search costs ignored in the evaluation metric. Deployed agents face a different regime: tasks arrive sequentially from a live stream with shifting domains and difficulties, making self-improvement inseparable from acting. We therefore study recursive self-improvement in an online learning formulation, where success is cumulative performance over the stream rather than the terminal performance of a single final variant (Cesa-Bianchi and Lugosi, 2006). From this online-learning view, each modification carries a cost, exploration is not free, and rewards observed at different times may not be directly comparable.

The Success-Story Algorithm (SSA) (Schmidhuber et al., 1997) is an important precursor for online self-modification. In this paper, we argue that the central challenge of self-modifying tree-

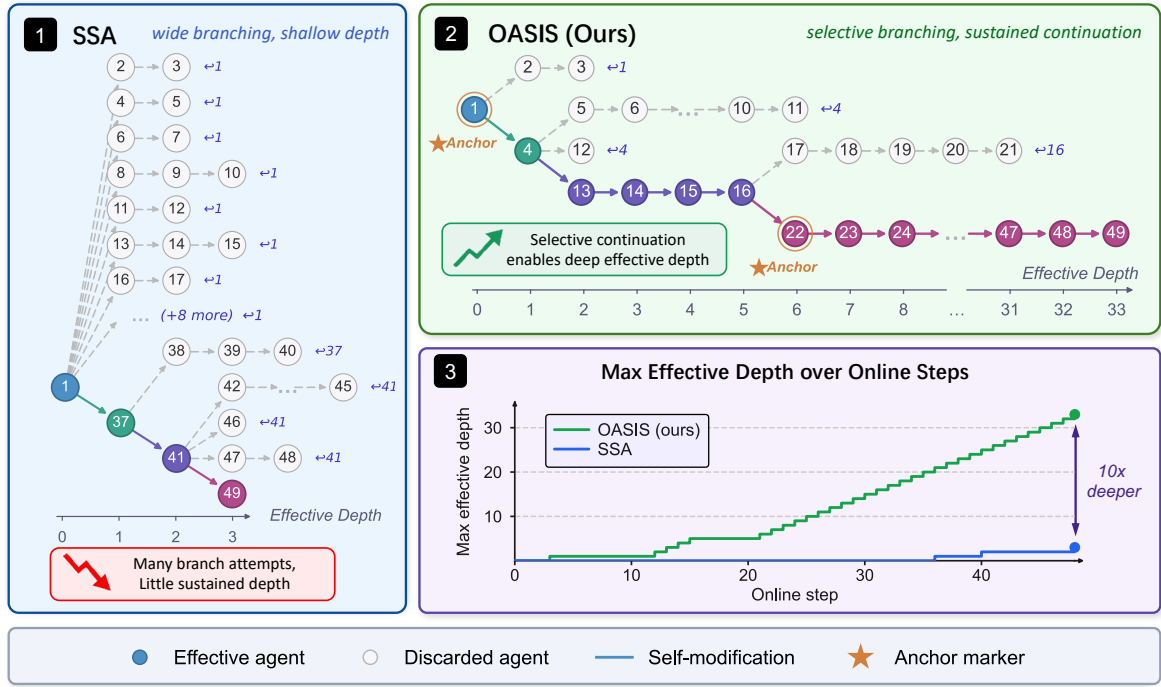


Figure 1: Comparison of agent search trees and effective depth progression between **SSA** and **OASIS (ours)** on the ANSIBLE benchmark from SWE-Bench Pro. **(1)** **SSA** repeatedly rolls back to early states, producing wide but shallow exploration. **(2)** **OASIS** selectively continues promising agents, sustaining a deep effective spine of 33 steps. **(3)** Max effective depth, i.e. the furthest depth reached along the final effective path, over online steps: OASIS retains a 10 $\times$ deeper spine than SSA with the same number of online steps. Nodes on the effective spine are shown in blue/purple; discarded branches in gray.

based systems is *comparability*: different lineages of self-modifications may be evaluated on different subsets of heterogeneous or nonstationary tasks. Hence, raw rewards are ambiguous: apparent rewards may reflect easier tasks rather than better agents. Yet SSA-style reward validation remains vulnerable to this ambiguity. For example, when later tasks are harder, it can become overly conservative and revert useful descendants. This yields a shallow effective depth with repetitive improvements and reversions instead of sustaining multi-step self-improvement. Figure 1 illustrates this failure mode of SSA and contrasts it with our anchor-based proposed approach.

To close this gap, we first introduce an online tree-based framework for recursive self-improvement under non-stationary task distributions. The framework maintains a growing tree of agent variants during deployment and must continually decide which agent should act, whether the current branch remains reliable, and which output to submit under limited evidence. Within this framework, we propose OASIS, an anchor-based approach for online recursive self-improvement. OASIS maintains an anchor agent that is evaluated

on every incoming task, independent of the active branch, and candidate agents or subtrees are compared against the anchor on the same tasks. This replaces raw branch-specific scores with paired comparisons, allowing the system to avoid shallow revert cycles and sustain deeper self-improvement trajectories, as illustrated in Figure 1. The anchor thus serves both as a validation instrument for confidence-based continuation, reversion, and promotion, and as a deployment safeguard when newly generated agents are suboptimal.

In summary, our contributions are threefold. First, we formulate recursive self-improvement as online learning over dynamic task streams, where agents modify themselves while acting and are evaluated by cumulative submitted reward. Second, we identify branch comparability as a central issue: descendants encounter different tasks, so absolute-reward validation can mistake task difficulty for agent quality, causing both excessive reversion and overly optimistic continuation in SSA. Third, we propose OASIS, an anchor-calibrated self-improvement method that compares branches against an adaptive reference, the anchor agent, on matched tasks, enabling confidence-calibrated con-

tinuation, reversion, promotion, and fallback, without exhaustive re-evaluation. Under identical initial scaffolds, foundation models, and streams, OASIS improves over both the initial agent and SSA on SWE-Bench Pro (Deng et al., 2025) and DeepResearch Bench (Du et al., 2025). Ablations show that anchor-calibrated branch control contributes more than best-of-two solution selection. Notably, OASIS achieves competitive performance compared to larger open weights and proprietary systems (with different scaffoldings) on SWE-Bench Pro and the best reported non-Deep-Research leaderboard result on DeepResearch Bench¹, while trajectory analyses show sustained gains rather than shallow revert cycles.

2 Online Recursive Self-Improvement

This section formalizes the problem of controlling a recursively self-improving agent for online learning problems as a learner growing a tree of agent variants. Then we state and analyze the Success Story Algorithm (Schmidhuber et al., 1997) under our formulation.

2.1 Problem Formulation

As in standard online learning (Littlestone and Warmuth, 1994; Cesa-Bianchi and Lugosi, 2006), tasks arrive sequentially, the system must produce an output before observing the feedback, and performance is measured by the cumulative reward of its submitted solutions over the task stream, rather than by the terminal performance of a final agent. Thus, the learner must simultaneously solve new tasks, expand the agent tree, and decide which agent is reliable enough to guide future behavior.

Formally, tasks arrive sequentially, $\tau_1, \tau_2, \dots, \tau_T$, where each task τ_t may come from a different domain and difficulty level. Each task is equipped with an evaluator $\text{Eval}_{\tau_t}(y) = (r, \xi)$, which returns a scalar reward r and, optionally, auxiliary feedback ξ , and is only available to the learner after the solution to the task is submitted. Note that this requirement of evaluation functions is a relaxation of supervised online learning, for which a ground truth target is required (Rosenblatt, 1958; Singh and Sharma, 2023). The online

objective is to maximize the cumulative reward of its submitted solutions over the stream.

Self-modification (by modifying the code, prompt, tools, memory, or control logic) generates a tree of agents. We denote the tree at time t as $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, with a node set $\mathcal{V}_t$ consisting of the existing agents and an edge set $\mathcal{E}_t$ consisting of the agent pairs (π, π') , where π' is a self-modification of π . The initial tree is $\mathcal{G}_0 = (\{\pi_0\}, \emptyset)$. We write $\mathcal{D}_t(\pi)$ for the descendants of π in $\mathcal{G}_t$, including π itself, and

$$\mathcal{P}_t(\pi) \triangleq (\pi_0, \pi_1, \dots, \pi_m = \pi)$$

for the ordered self-modification path from the root to π , where each π_{i+1} is a child of π_i .

A time step proceeds as follows. First, the system applies a set of agents, including an active agent π_t^{act} with $\pi_0^{\text{act}} = \pi_0$, to the incoming task. Let $\mathcal{Q}_t$ denote this set. Each agent $\pi \in \mathcal{Q}_t$ produces a candidate solution $y_{t,\pi} = \pi(\tau_t)$. If more than one solution is present, the system must choose one solution $y_t \in \{y_{t,\pi} : \pi \in \mathcal{Q}_t\}$ to submit as the system’s attempted answer to task τ_t . As in the standard online learning paradigm, only r_t , the reward of the submitted solution y_t , contributes to the online learning objective: $\sum_{t=1}^T r_t$.

After submission, the task evaluator becomes available. We then store all task-solving records from round t . More precisely, for any agent π , let $\mathcal{H}_t(\pi)$ multiset denote the records associated with π before round t with $\mathcal{H}_1(\pi) \triangleq \emptyset$ and

$$\mathcal{H}_{t+1}(\pi) \triangleq \mathcal{H}_t(\pi) \cup \{e_{t,\pi} : \pi = \pi', \pi' \in \mathcal{Q}_t\},$$

where $e_{t,\pi} = (\pi, \tau_t, \text{Eval}_{\tau_t}(y_{t,\pi}))$. Given a record e , we write $r(e)$ for the reward stored in that record. For any node $\pi \in \mathcal{V}_t$, we define a subtree-level evidence set

$$\mathcal{R}_t^{\text{sub}}(\pi) \triangleq \{r(e) : \pi' \in \mathcal{D}_t(\pi), e \in \mathcal{H}_t(\pi')\} \quad (1)$$

containing the raw rewards obtained by agents in the subtree rooted at π . This descendant-level view is natural for recursive self-improvement: a modification should be judged not only by the immediate performance of the agent it creates but also by the quality of the future agents reachable from it. This is aligned with the descendant-aware perspective of the optimal self-improvement machine—Gödel Machine (Schmidhuber, 2003), which is highlighted by Wang et al. (2026) in the mismatch between immediate task performance and the ability to generate strong descendants.

¹<https://huggingface.co/spaces/muset-ai/DeepResearch-Bench-Leaderboard> accessed on May 25, 2026. We refer to Non-Deep-Research as submissions that are not labeled as Deep Research Agents and do not include “research” in the method name.

Given the currently available information, the active agent may then produce a descendant by proposing a self-modification:

$$\tilde{\pi}_t \leftarrow \text{SelfModify}(\pi_t^{\text{act}}; \mathcal{H}_{t+1}(\pi_t^{\text{act}}), \{\text{Eval}_{\tau'_t} : t' \in \{1, 2, \dots, t\}\}),$$

where $\tilde{\pi}_t = \emptyset$ if no modification is proposed. If $\tilde{\pi}_t \neq \emptyset$, it is added to the tree as a child of π_t^{act} .

Finally, at the end of each round, the system updates its active pointer. If a child $\tilde{\pi}_t$ is produced, it becomes the provisional active agent for the next round; otherwise, the provisional active agent remains π_t^{act} . A reversion rule may then move this pointer back to an ancestor on the same modification path: $\pi_{t+1}^{\text{act}} \in \mathcal{P}_{t+1}(\tilde{\pi}_{t+1})$, where $\tilde{\pi}_{t+1}$ denotes the provisional active agent after self-modification. Thus, online recursive self-improvement requires more than generating new agents: the system must decide which branch should continue, when to revert, which solution to submit, and how to compare agents evaluated on different parts of a changing task stream. The next subsection shows why raw subtree rewards are insufficient for these decisions.

2.2 Success-Story Algorithm for Online RSI

The closest existing method applicable to our online RSI formulation is the Success-Story Algorithm (SSA) (Schmidhuber et al., 1997), which was introduced for a more general single-life reinforcement learning setting (Schmidhuber, 1994; Chen et al., 2022). In the notation of the previous subsection, SSA corresponds to a special case in which only the active agent is run on each task: $\mathcal{Q}_t = \{\pi_t^{\text{act}}\}$. Thus, the reward at time t is simply the reward of the current active agent’s generated solution, and the system evaluates self-modifications using the single interaction history generated by the active trajectory.

SSA captures a natural validation principle: a self-modification should be retained only if the future interaction history following that modification improves over the histories that preceded it. In our agent-tree formulation, the SSA requirement can be framed as a monotonicity condition along the active path $\mathcal{P}_t(\pi_t^{\text{act}})$:

$$\rho_t^{\text{raw}}(\pi_0) < \rho_t^{\text{raw}}(\pi_1) < \dots < \rho_t^{\text{raw}}(\pi_m),$$

where $\rho_t^{\text{raw}}(\pi) = \frac{\sum_{r \in \mathcal{R}_t^{\text{sub}}(\pi)} r}{|\mathcal{R}_t^{\text{sub}}(\pi)|}$. If this condition fails, an SSA-style validator may interpret the later modification as unsuccessful and revert the active pointer to an earlier node on the path.

A limitation of comparing these raw subtree scores is that these scores are neither task-matched nor uncertainty-aware. The evidence scores used to compare two nodes come from different portions of the online task stream, which may differ in domain composition and difficulty. A higher raw subtree score may therefore reflect easier task exposure rather than a better agent, while a lower score may reflect a harder segment of the stream rather than a harmful modification. Moreover, newly created descendants often have few run records, making the SSA monotonicity condition high-variance and sensitive to stochasticity. We therefore desire a validation signal that preserves the online nature of SSA while replacing the absolute reward criterion with a task-matched and uncertainty-aware criterion. The next section introduces OASIS, an anchor-based method that provides such comparisons without exhaustively evaluating every agent on every task.

3 Online Anchor-calibrated Self-Improvement System

Section 2 shows that absolute rewards are not a reliable criterion for online recursive self-improvement; e.g., a tree branch that receives a low reward may be weak, or it may simply have faced harder tasks. This motivates our approach, which involves validating decisions relative to a reference agent evaluated on the same tasks. We call this reference the *anchor agent*.

The anchor serves as the calibration point for OASIS. First, it converts raw rewards into task-matched comparisons: instead of asking whether a branch obtained a high reward, we ask whether it outperformed the anchor on the same tasks. Second, it provides a conservative deployment fallback: when a newly generated agent has few evaluations and is potentially weak, the system can continue exploring while submitting the anchor’s solution.

3.1 The Anchor as a Task-Matched Reference

At each round t , our approach maintains an anchor agent $\pi_t^{\text{anc}} \in \mathcal{Q}_t$ and includes it in the set of agents run on the incoming task. Thus, every task is associated with a reference reward from the anchor. This is the key intervention: each observed reward from a candidate branch is paired with the reward of the anchor on the same task. Thus, whenever a branch accumulates evidence on a task, our method also obtains an anchor reward for that task. This

gives each branch a matched reference history. Our framework also allows the anchor to improve over time: when a descendant is validated as better than the current anchor, we promote it and use it as the reference for future tasks.

For any branch rooted at π , the raw subtree reward set $\mathcal{R}_t^{\text{sub}}(\pi)$, defined in Eq. (1), collects the rewards obtained by the descendants of π . We define the corresponding anchor-based reward multiset as

$$\mathcal{R}_t^{\text{anc}}(\pi) \triangleq \left\{ r_{t', \pi_t^{\text{anc}}} : \pi' \in \mathcal{D}_t(\pi), e_{t', \pi'} \in \mathcal{H}_t(\pi') \right\}, \quad (2)$$

where $r_{t', \pi_t^{\text{anc}}}$ is the reward produced by the anchor at the time task $\tau_{t'}$ was arrived. That is, for each run record evaluating the subtree rooted at π , we include the contemporaneous anchor reward for the same task. Thus, $\mathcal{R}_t^{\text{sub}}(\pi)$ and $\mathcal{R}_t^{\text{anc}}(\pi)$ share task exposure: the former records branch performance, while the latter provides the anchor reference.

Intuitively, a raw subtree score asks whether a branch performed well on the tasks it happened to encounter. Our anchor-based comparison asks whether that branch performed better than a shared reference on the same task exposure. This removes the main ambiguity in dynamic task streams: changes in task difficulty or domain composition are no longer interpreted directly as changes in agent quality. At the same time, the method remains online and computationally feasible because we only apply the anchor agent to the tasks in addition to the active agent and its self-modification, rather than evaluating every agent on every task.

The anchor also provides a deployment safeguard. Since it is evaluated throughout the task stream, it accumulates a broad and continuously updated performance record. When the active branch is newly generated or statistically under-evaluated, the system can continue exploring it while using the anchor as a conservative fallback for submission.

3.2 Confidence-Calibrated Validation

The anchor-based reward sets make branch comparisons task-matched, but finite-sample uncertainty remains. A branch may outperform the anchor on a few tasks by chance. We therefore compare posterior credible intervals for the expected rewards of the branch and anchor under the same task exposure.

For every tree node π , let $S^{\text{sub}}(\pi)$ and $S^{\text{anc}}(\pi)$ denote the independent normalized reward of its subtree and matched anchor, respectively. For an observation multiset $D = \{r_i\}_{i=1}^n$ with $r_i \in [0, 1]$,

the implementation uses the fractional-count update

$$S \mid D \sim \text{Beta} \left(2 + \sum_i r_i, 2 + n - \sum_i r_i \right). \quad (3)$$

Let $F_{S \mid D}$ be the posterior cumulative distribution function. For credible level $\alpha \in (0, 1)$, we define

$$\text{LCB}_\alpha(S \mid D) = F_{S \mid D}^{-1} \left(\frac{1 - \alpha}{2} \right), \quad (4)$$

$$\text{UCB}_\alpha(S \mid D) = F_{S \mid D}^{-1} \left(\frac{1 + \alpha}{2} \right). \quad (5)$$

Hence $[\text{LCB}_\alpha, \text{UCB}_\alpha]$ is the central α -level posterior credible interval. We act only when the subtree and anchor intervals separate.

3.3 Tree-Control and Solution Selection Decisions

The same confidence logic governs the three decisions needed for online self-improvement: whether to continue a branch, when to update the anchor, and which solution to submit.

Reversion. We revert a branch only when it is confidently worse than the anchor, conditioned on the same task exposure:

$$\text{UCB}_{\alpha_1}(S \mid \mathcal{R}_t^{\text{sub}}(\pi)) < \text{LCB}_{\alpha_1}(S \mid \mathcal{R}_t^{\text{anc}}(\pi)). \quad (6)$$

For some $\alpha_1 \in (0, 1)$, along the provisional root-to-active path, we identify the earliest non-root node satisfying Eq. (6) and revert to its parent. If no node qualifies, the provisional active agent is retained. This rule avoids reverting a modification merely because it encounters harder tasks or because its early evaluations are noisy.

Anchor promotion. We promote a descendant on the current active lineage as the new anchor only when it is confidently better than the current anchor on the same task exposure:

$$\text{LCB}_{\alpha_2}(S \mid \mathcal{R}_t^{\text{sub}}(\pi)) > \text{UCB}_{\alpha_2}(S \mid \mathcal{R}_t^{\text{anc}}(\pi)). \quad (7)$$

For some $\alpha_2 \in (0, 1)$, if multiple nodes qualify, we promote the deepest one. The promoted node becomes both the anchor and active agent, perserving the matched anchor-active recored reward match. Promotion is checked before reversion. Thus, both decisions use matched evidence: branches are demoted only with evidence of inferiority, and anchors are replaced only with evidence of superiority.

Solution Selection. For each task at time t , both the active agent and the anchor agent produce candidate solutions. We use a large language model (LLM) judge (Zheng et al., 2023) to choose between them.

4 Related Works

LLM Agents for Online Learning. Online learning evaluates systems on task streams by cumulative loss or regret rather than by terminal performance on a fixed test set (Cesa-Bianchi and Lugosi, 2006). For LLM agents, this perspective has mostly been explored empirically. A common strategy is to accumulate task experience in external memory queried at inference time, by compounding knowledge over time (Wang et al., 2023), extracting and storing reusable insights (Zhao et al., 2024), or reusing common workflows (Wang et al., 2025). In these systems, however, updates are a limited fixed component of the agent and follow predefined rules. We study a broader setting in which agents recursively self-modify across a larger and more flexible design space with less predictable update policies, making robustness and fallback safeguards central methodological concerns. Alternatively, Xia et al. (2025) removes the offline training phase and allows agents to synthesize tools at runtime while solving an issue. Although this self-evolution occurs at deployment time, it remains confined to a single issue: created tools are discarded at episode boundaries, the agent is reset to its initial scaffold for each new task, and evaluation remains the standard per-issue solve rate.

Matched Online Evaluation. Contemporaneous control arms and online controlled experiments compare alternatives under shared conditions to reduce confounding from temporal changes (Kohavi et al., 2009). Dueling-bandit methods likewise learn from pairwise comparisons between actions rather than absolute rewards (Yue et al., 2012). OASIS uses the same broad principle of matched evidence, but for a different control problem: the reference is adaptive and promotable, and the comparisons determine which branch of a growing self-modification tree should continue, revert, become the new anchor, and supply a deployable fallback.

Recursive Self-Improvement. The idea of recursive self-improvement, where a system improves not only its task performance but also the very mechanism by which it improves itself,

dates back to Good (1966) and was first concretized algorithmically by Schmidhuber (1987). Follow-up works, including the Success-Story Algorithm (Schmidhuber et al., 1997) and Gödel machines (Schmidhuber, 2003), study a scenario where a system interacts with its environment over a single reinforcement learning (RL) trajectory and accepts self-modifications based on long-term reward acceleration, with no separation between training and deployment. This RL formulation is more general than our online learning setting since it does not require task separation. Recent LLM-based self-improving systems, including STOP (Zelikman et al., 2024), Gödel Agent (Yin et al., 2025), Darwin-Gödel Machine (Zhang et al., 2026a), Huxley-Gödel Machine (Wang et al., 2026), SEAL (Zweiger et al., 2026), and Absolute Zero (Zhao et al., 2026), have largely departed from this dynamic setting and have instead adopted an *offline* self-improvement paradigm in which systems are optimized within a closed search loop, and the resulting frozen artifact is evaluated on static benchmarks. In contrast, we study recursive self-improvement in an online setting over a task stream, evaluated by cumulative performance during the online learning process.

5 Empirical Results

We evaluate OASIS as an online RSI method for agentic systems operating over streaming task sequences. Our experiments answer two questions. First, in Section 5.1, does OASIS improve cumulative task performance compared with both a fixed initial agent and the prior SSA baseline? Second, in Section 5.2, does anchor-calibrated validation produce more effective and progressive self-improvement trajectories under nonstationary task distributions and noisy feedback? We test our method in coding and research using the SWE-Bench Pro (Deng et al., 2025) and DeepResearch Bench (Du et al., 2025) benchmarks.

Setup: OASIS is a tree-control and validation framework that is agnostic to the representation of a self-modification. Our experiments instantiate it with Claude Code skills (Anthropic, 2026). A skill can contain instructions, reusable procedures, and arbitrary code or scripts, so this representation can implement changes to prompts, tools, memory, and control logic rather than only textual prompting. Starting from the same initial agent with no predefined skills, each self-modification adds or revises

skills after an active-agent task-solving attempt and its evaluation. The SWE-Bench Pro experiments use Qwen3.6-35B-A3B (Qwen Team, 2026); the DeepResearch Bench experiments use Gemini 3 Flash (Doshi and Team, 2025) with the RACE metric (Du et al., 2025) reported. Since OASIS selects between active- and anchor-agent solutions, the non-adaptive initial-agent baseline also generates two solutions and uses the same LLM judge. This practice belongs to the common paradigm known as best-of-k, a typical way of test-time scaling to improve agentic solutions (Stiennon et al., 2020; Chen et al., 2021). To further disentangle the sources of the gain, we perform an ablation on SWE-Bench Pro that evaluates a compute-matched SSA variant that selects between the active SSA agent and the initial agent with the same judge. This setting isolates the contributions of anchor-calibrated branch control and test-time selection while controlling the number of task-solving roll-outs.

We construct task streams for SWE-Bench Pro and DeepResearch Bench by grouping SWE-Bench Pro tasks by Git repository in a randomly permuted order and grouping DeepResearch Bench tasks by language, with topics ordered alphabetically. This ordering is chosen to avoid introducing prior assumptions and to model sequential topic-distribution shifts within each stream, respectively. In total, our experiments cover 11 SWE-Bench Pro streams, each consisting of 20 to 29 sequential steps with a batch of tasks arriving at each step, and two DeepResearch streams, each consisting of 25 such steps. See Appendix C.1 for more detail.

In the experiments, we assume the subtree normalized rewards follow $\text{Beta}(2, 2)$ mutually independently. We use $\alpha_1 = 0.4$ for reversion and $\alpha_2 = 0.6$ for promotion. The wider promotion interval intentionally requires stronger evidence to replace the anchor. A sensitivity run with $\alpha_1 = \alpha_2 = 0.95$ reaches 53.9% on SWE-Bench Pro, compared with 56.9% for the reported setting. To improve wall-clock efficiency and stability, OASIS and SSA generate self-modifications at the end of each batch. We refer to Appendix C for additional details.

5.1 OASIS Online Learning Effectiveness

Results & Discussion. Table 1 and Table 2 summarize the main comparison results in average performance across tasks in all batches and streams. On SWE-Bench Pro, the fixed initial agent achieves

51.6%, SSA reaches 52.0%, and compute-matched SSA reaches 52.5%. In contrast, OASIS reaches 56.9%, improving over the initial agent by 5.3 points and over compute-matched SSA by 4.4 points. Removing best-of-two selection reduces OASIS by 1.1 points to 55.8%, whereas replacing its branch control with the SSA rule reduces it by 4.4 points to 52.5%. Thus, for SWE-Bench Pro, both components contribute, with the larger measured contribution coming from anchor-calibrated branch control. On DeepResearch Bench, the initial agent achieves 35.18, SSA improves to 37.03, and OASIS reaches 39.31, yielding gains of 4.13 points over the initial agent and 2.28 points over SSA.

The absolute performance of OASIS is also notable. At the time of writing, its 56.9% SWE-Bench Pro solve rate places it among the strongest open-weight-agent results. With Qwen3.6-35B-A3B, a model with 35 billion parameters and 3 billion active parameters per token generation, it outperforms systems built on substantially larger foundation models, such as MiniMax M2.7 and Qwen3.5-397B-A17B, and also marginally outperforms systems with proprietary models such as GPT-5.3-Codex and Qwen3.6-Plus, with potentially different scallfoldings. On DeepResearch Bench, its score of 39.31, surpasses the best-reported non-Deep-Research result on the leaderboard¹. These results suggest that online self-improvement can enhance strong agentic systems in both coding and research.

5.2 Analysis of Self-Improvement Trajectories

We next analyze the self-improvement trajectories induced by each online RSI method on the 11 SWE-Bench Pro streams. Instead of measuring only the aggregate solve rate, we characterize whether self-modifications form sustained effective lineages and yield downstream benefits. Figure 2 presents the average maximum effective depth over online steps, where the effective depth is measured from the initial root and denotes the longest consecutive chain of modifications that remains on the final retained path. The quantity captures the persistence modifications made up to each step rather than a quantity available to the agent at that time. Because this quantity is directly affected by each method’s reversion policy, it is a trajectory diagnostic rather than an independent measure of agent quality. Performance in Table 1 remains the primary evidence. Furthermore, Figure 3 shows the average future

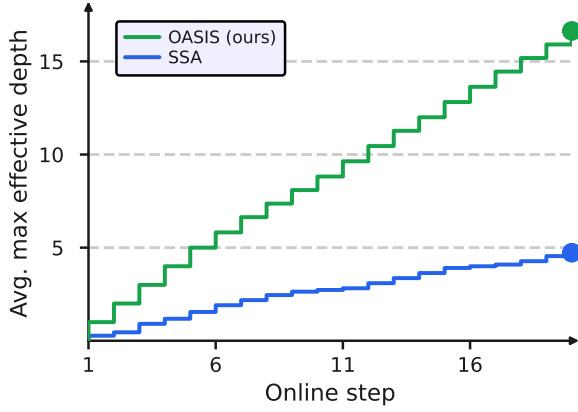


Figure 2: **Progressive Average Max Effective Depth over Online Steps on SWE-Bench Pro.** OASIS reaches substantially deeper effective self-improvement trajectories than SSA, indicating sustained continuation rather than shallow branching.

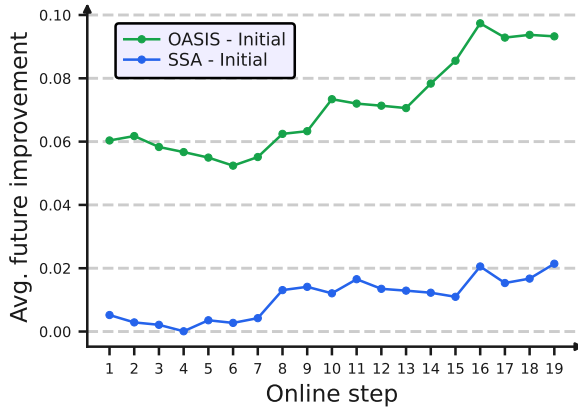


Figure 3: **Progressive Average Future Improvement on SWE-Bench Pro.** The curves show the mean stream-level solve-rate gain over the non-adaptive initial agent on batches following each online step.

improvement of each agent over the non-adaptive initial agent at each online step, measured by its solve-rate difference on subsequent batches across the 11 repository streams. Unlike the overall task-average in Table 1, this per-stream suffix metric directly reflects the online objective at each checkpoint.

Together, these analyses test whether online self-modification produces compounding improvements rather than isolated short-term gains. Following our hypothesis, SSA’s reward-based validation tends toward shallow effective depth and limited future utility, whereas the anchor-calibrated validation in OASIS yields deeper effective trajectories and increasingly beneficial future performance.

Results & Discussion. Figure 2 shows that OASIS produces substantially deeper and superior self-modifications than SSA. In terms of maximum

effective depth, OASIS steadily increases throughout the online process, whereas SSA remains shallow, indicating that OASIS often preserves modifications that are preferred in the future. Figure 3 further reinforces this conclusion: OASIS maintains a consistently larger future solve-rate advantage over the fixed initial agent, and this advantage grows as online learning progresses, while SSA yields only limited downstream gains.

For a formal comparison, at each of the 19 checkpoints we compute the paired OASIS-minus-SSA future-performance difference within each stream, assuming the streams are independently sampled. We exhaustively enumerate all 2^{11} stream-level sign flips for a one-sided test, applying one sign to a stream’s complete checkpoint vector so that dependence from nested task suffixes is preserved. A Westfall–Young min- P adjustment controls family-wise error across checkpoints. The global intersection null of no positive OASIS advantage at any checkpoint is rejected ($p = 0.031$). The Hodges–Lehmann estimate is positive at every checkpoint, while steps 16–18 are individually significant after adjustment (adjusted $p = 0.031$ – 0.037).

Together, these results suggest that OASIS does not merely discover isolated improvements, but accumulates modifications that remain useful for subsequent tasks and compound over time. The contrast with SSA indicates that anchor-calibrated validation improves the reliability of online self-modification by supporting sustained effective lineages and selecting agent states with greater future utility.

6 Conclusion

This paper formulated recursive self-improvement in the setting where it matters most for deployed agents: online interaction with a nonstationary stream of tasks. We argued that this setting differs fundamentally from offline agent discovery. In deployment, self-improvement is not judged by the best final variant found after the search, but by cumulative performance while the system is acting, modifying itself, and deciding which agent variant to trust on the fly. Our central observation is that online recursive self-improvement is challenging in terms of task comparability. Because different agent variants encounter different tasks, raw rewards can confound agent capability with task difficulty, causing reward-based validation to spuriously reject useful modifications or

Table 1: **SWE-Bench Pro solve rate.** Our method with Qwen3.6-35B-A3B achieves a 56.9% solve rate, clearly outperforming SSA and improving substantially over the initial agent. Despite using a significantly smaller parameter count, our result surpasses much larger open-weight models and several recent proprietary models.

Method / Model	Score (%)
<i>Qwen3.6-35B-A3B</i>	
OASIS (full)	56.9 ^{+5.3}
OASIS w/o best-of-two	55.8 ^{+4.2}
OASIS ($\alpha_1=\alpha_2=0.95$)	53.9 ^{+2.3}
SSA with best-of-two	52.5 ^{+0.9}
SSA	52.0 ^{+0.4}
Initial with best-of-two	51.6 ^{ref.}
<i>Systems with Open-Weight Models</i>	
MiniMax M2.7 (230B-A10B)	56.2
Qwen3.5-397B-A17B	50.9
<i>Systems with Proprietary Models</i>	
GPT-5.3-Codex	56.8
Qwen3.6-Plus	56.6

Note: Superscripts are absolute improvements over the matched Initial reference.

retain weak ones. To address this, we proposed OASIS, an anchor-calibrated approach that compares candidate branches against an adaptive reference agent on matched task exposures. This enables confidence-calibrated continuation and fallback output selection without exhaustive evaluations. Across streaming SWE-Bench Pro and DeepResearch Bench, OASIS improves upon both the initial agent and SSA under matched conditions, and trajectory analyses show sustained gains rather than shallow revert cycles. Our SWE-Bench Pro ablation attributes most of the measured gain to branch control rather than best-of-two selection.

More broadly, this work reframes recursive self-improvement as an online learning problem with an evolving decision class. We believe this perspective opens a path toward self-improving agents that can adapt during deployment while controlling the risks of noisy feedback, changing task difficulty, and premature commitment to unreliable descendants.

Limitations

OASIS assumes a relatively structured online environment. Unlike single-life RL settings considered by SSA and Gödel-machine-style agents, our formulation assumes that tasks are identifiable and evaluations are reproducible. Extending anchor-

Table 2: **DeepResearch Bench scores.** Our optimized system achieves 39.31, substantially improving over the initial agent, SSA, and the best non-Deep-Research result reported on the leaderboard¹.

Method	Score
<i>Gemini 3 Flash</i>	
OASIS	39.31 ^{+4.13}
SSA	37.03 ^{+1.85}
Initial	35.18 ^{ref.}
Best Non-Deep-Research LB	38.22 ¹

calibrated validation to less structured continuous environments remains an open problem.

The method also depends on usable feedback. Automatic or semi-automatic rewards make task-level comparisons practical, but settings with sparse, delayed, subjective, or multi-objective rewards may provide weaker calibration signals. In such cases, measured improvements may only partially capture usefulness, robustness, or safety.

Our evaluation horizon is limited. Longer runs would better test whether anchor-calibrated validation continues to compound improvements, avoids late-stage regressions, and remains robust as feedback accumulates over many successive self-modifications. OASIS also adds inference costs by maintaining an anchor agent, creating a deployment-dependent trade-off between reliability, exploration, and budget.

Acknowledgment

We thank Mateusz Ostaszewski for the discussion of related works. This material is based upon work supported by the Google Cloud Research Credits program with the award GCP19980904 and by funding from King Abdullah University of Science and Technology (KAUST) - Center of Excellence for Generative AI, under award number 5940.

References

- Anthropic. 2026. [Claude code](#).
- Nicolo Cesa-Bianchi and Gábor Lugosi. 2006. *Prediction, learning, and games*. Cambridge university press.
- Annie Chen, Archit Sharma, Sergey Levine, and Chelsea Finn. 2022. You only live once: Single-life reinforcement learning. *Advances in Neural Information Processing Systems*, 35:14784–14797.

- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Naveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.
- Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. 2025. [Swe-bench pro: Can ai agents solve long-horizon software engineering tasks?](#) *CoRR*, abs/2509.16941.
- Tulsee Doshi and Gemini Team. 2025. Gemini 3 flash: frontier intelligence built for speed. <https://blog.google/products-and-platforms/products/gemini/gemini-3-flash/>. Google Blog, The Keyword. Accessed: 2026-05-26.
- Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. 2025. Deepresearch bench: A comprehensive benchmark for deep research agents. *arXiv preprint arXiv:2506.11763*.
- Irving John Good. 1966. Speculations concerning the first ultraintelligent machine. In *Advances in computers*, volume 6, pages 31–88. Elsevier.
- Shengran Hu, Cong Lu, and Jeff Clune. 2024. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*.
- Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, and 1 others. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.
- Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal M Henne. 2009. Controlled experiments on the web: survey and practical guide. *Data mining and knowledge discovery*, 18(1):140–181.
- Nick Littlestone and Manfred K Warmuth. 1994. The weighted majority algorithm. *Information and computation*, 108(2):212–261.
- Qwen Team. 2026. [Qwen3.6-35b-a3b](#).
- Maxime Robeyns, Martin Szummer, and Laurence Aitchison. 2025. [A self-improving coding agent](#). *Preprint*, arXiv:2504.15228.
- Frank Rosenblatt. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.
- Juergen Schmidhuber. 1994. [On learning how to learn learning strategies](#).
- Jürgen Schmidhuber. 1987. Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook. Diploma thesis, Institut für Informatik, Technische Universität München.
- Jürgen Schmidhuber. 2003. Gödel machines: self-referential universal problem solvers making provably optimal self-improvements. *arXiv preprint cs/0309048*.
- Jürgen Schmidhuber, Jieyu Zhao, and Marco Wiering. 1997. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 28(1):105–130.
- Charanjeet Singh and Anuj Sharma. 2023. A review of online supervised learning. *Evolving Systems*, 14(2):343–364.
- Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021.
- Alan M. Turing. 1950. [Computing machinery and intelligence](#). *Mind*, LIX(236):433–460.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. [Voyager: An open-ended embodied agent with large language models](#). *Preprint*, arXiv:2305.16291.
- Wenyi Wang, Piotr Piękos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. 2026. [Huxley-Gödel machine: Human-level coding agent development by an approximation of the optimal self-improving machine](#). In *International Conference on Learning Representations (ICLR)*. Oral.
- Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025. [Agent workflow memory](#). In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 63897–63911.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang,

- Shaokun Zhang, Jiale Liu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*.
- Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. 2025. [Live-SWE-agent: Can software engineering agents self-evolve on the fly?](#) *arXiv preprint arXiv:2511.13646*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.
- Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. 2025. [Gödel agent: A self-referential agent framework for recursive self-improvement](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL): Long Papers*.
- Yisong Yue, Josef Broder, Robert Kleinberg, and Thorsten Joachims. 2012. The k-armed dueling bandits problem. *Journal of Computer and System Sciences*, 78(5):1538–1556.
- Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. 2024. [Self-taught optimizer \(STOP\): Recursively self-improving code generation](#). In *Conference on Language Modeling (COLM)*. Spotlight.
- Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. 2026a. [Darwin Gödel machine: Open-ended evolution of self-improving agents](#). In *International Conference on Learning Representations (ICLR)*.
- Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. 2026b. Hyperagents. *arXiv preprint arXiv:2603.19461*.
- Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.
- Andrew Zhao, Yiran Wu, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. 2026. Absolute zero: Reinforced self-play reasoning with zero data. *Advances in Neural Information Processing Systems*, 38:105816–105879.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhaghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.
- Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning, ICML’24*. JMLR.org.
- Adam Zweiger, Jyo Pari, Han Guo, Yoon Kim, and Pulkit Agrawal. 2026. Self-adapting language models. *Advances in Neural Information Processing Systems*, 38:74084–74115.

A Potential Risks

Self-improving agents pose risks beyond static language-model agents because they can change the procedures that shape future behavior. Repeated modifications may produce unexpected tool use, reward hacking, degraded robustness, or policies that optimize measured reward while violating user intent. Anchor calibration is a reliability mechanism, not a complete safety guarantee. If the evaluator misses harmful behavior, or if rare failure modes are absent from the task stream, OASIS may still validate an undesirable modification.

Misuse risks include automating vulnerability discovery, generating insecure code at scale, optimizing deceptive writing or research workflows, and adapting agents to bypass operational constraints. Deployments should therefore use sandboxing, explicit permission boundaries, audit logs, regression and safety tests, and human reviews for high-impact changes before promotion. In high-stakes domains, self-modification should be restricted or disabled until stronger oversight and guarantees are available.

B Use or Create Scientific Artifacts

We use existing scientific artifacts, including SWE-Bench Pro (Deng et al., 2025), DeepResearch Bench (Du et al., 2025), their associated benchmark evaluation harnesses, foundation models (Qwen3.6-35B-A3B (Qwen Team, 2026), Gemini 3 Flash (Doshi and Team, 2025), and Gemini 2.5 Pro (Comanici et al., 2025)), and Claude Code as the agent execution environment (Anthropic, 2026). We use these artifacts solely for research evaluation and comply with their stated access conditions, licenses, and intended-use restrictions. DeepResearch Bench is released under the Apache 2.0 License. SWE-Bench Pro tasks and codebases are derived from upstream open-source software repositories, including strong copyleft repositories, so the corresponding code and task content should be

treated as subject to the original upstream repository licenses rather than as uniformly permissive material. Foundation models and proprietary services are used under their provider access terms; we do not redistribute model weights, proprietary service implementations, or third-party benchmark or code content in ways that conflict with those terms.

We also create experimental artifacts, including task streams, learning pipelines, and performance results. The accompanying release includes the code, exact task-solving and self-modification prompts, materialized streams, configurations, intermediate agent trees, and history records needed to resume experiments from an intermediate point of a stream. We do not intentionally collect new personal data or introduce offensive content. The benchmark tasks may contain repository text, issue text, documents, or other content inherited from their source artifacts. We use this content solely for the original benchmark tasks and do not use the resulting logs as a general-purpose dataset of user interactions.

C Implementation Details

C.1 Stream Construction

We evaluate online self-improvement on fixed, materialized task streams that are replayed deterministically across all comparisons. SWE-Bench Pro is partitioned by source repository, giving eleven repository-specific streams drawn from the ScaleAI SWE-bench Pro test split; DeepResearch Bench is partitioned by language, giving an English stream and a Chinese stream drawn from the benchmark tasks. Within each SWE-Bench Pro stream, we apply a random permutation to the per-repository task list since our goal is to model the deployment-time arrival of issues rather than any particular semantic ordering. Within each DeepResearch stream, we order tasks by the topic field provided by the dataset, which induces a smooth topic-level distribution shift along the stream. We plan to release the resulting materialized streams alongside our codebase.

Each stream is then divided into consecutive batches. We target thirty nominal batches per stream and set the batch size to $\lceil N/30 \rceil$, where N is the number of tasks in the stream, yielding batch sizes between one and four. Table 3 reports the stream lengths and batch sizes used in the reported runs.

Stream	Tasks	Batch size
Ansible	96	4
OpenLibrary	91	4
Flitpt	85	3
Qutebrowser	79	3
Teleport	76	3
ProtonMail	65	3
Vuls	62	3
Navidrome	57	2
Element Web	56	2
NodeBB	44	2
Tutanota	20	1
DeepResearch-en	50	2
DeepResearch-zh	50	2

Table 3: Materialized stream lengths and batch sizes for the reported runs.

C.2 Task Solving

At each online step, OASIS exposes the incoming batch of tasks to two agents: the current active agent and the current anchor agent. Each agent runs inside its own Docker container so that filesystem edits, evaluation artifacts, and intermediate logs of one agent cannot interfere with those of the other. Agent state is represented as a set of reusable skills that the in-container coding harness selectively activates while solving a task. Before solving begins, the runner installs the harness, transfers the agent’s skills into the container, configures version control, and records the starting state of the task repository. The agent is then invoked in the non-interactive mode of Claude Code with the benchmark task instruction followed by a short continuation reminder asking the agent to keep working until the task is complete and to refrain from soliciting clarification.

For SWE-Bench Pro, each task uses the benchmark’s prebuilt container image with a pre-setup script and operates in a per-task working directory inside that container. Once the agent finishes, the resulting patch is captured as the diff from the recorded base commit, and any modifications to protected test files are reverted prior to grading. The evaluator runs test commands provided by the benchmark and parses the resulting test log. A task is marked as solved, following the benchmark evaluation protocol, only when every fail-to-pass test passes and no pass-to-pass test regresses.

For DeepResearch Bench, the runner brings up a container image prebuilt from a slim Python image with version control, network utilities, and certificate authorities pre-installed. Inside the container, the runner initializes a clean working directory, writes the task metadata as a structured manifest,

and emits an output specification that lists the files the agent is expected to produce. The agent is asked to write a Markdown research report with checkable source URLs alongside a structured JSON summary of the article; the English and Chinese prompts mirror each other in their respective languages, and both explicitly forbid the agent from implementing its own evaluator or scorer and from embedding secrets in the outputs. Following the benchmark rubric, a report is considered format-valid only when it is non-empty, contains at least five hundred characters, and includes at least one URL. Valid reports are then scored by the DeepResearch RACE judge against the benchmark criteria and the reference article, and the resulting normalized overall score is used as the task reward.

When both agents return a candidate solution, the submission is chosen by an LLM judge following the LLM-as-a-judge protocol of [Zheng et al. \(2023\)](#). To keep the online reward strictly comparable across methods, we extend this best-of-two test-time selection to the non-adaptive initial-agent baseline by running it twice on every task and applying the same LLM judge to its two candidates.

C.3 Self-Modification

Self-modification is performed once the task evaluation function is available and runs once per task in the current batch. Self-modification is performed in the same container used for task solving, so it can inspect the task repository, the agent’s full output, the evaluator feedback, and any test logs in their original on-disk form. Claude Code is re-entered in continuation mode so that the agent reasoning begins from the active task-solving trajectory rather than from a blank context.

The self-modification prompt opens with an explicit phase boundary stating that task solving is over and that task files under the benchmark working directory must not be edited; it then directs the agent to create or refine reusable skills under the agent’s skills directory. When the task was solved, the prompt asks the agent to analyze what worked and to encode that knowledge as a reusable skill; when the task was not solved, the prompt includes the evaluator feedback and asks the agent to encode the missing knowledge or tooling, while explicitly forbidding any attempt to retry the task itself. In both branches, the prompt prohibits task-specific facts, file names, issue identifiers, and one-off fixes, and emphasizes skills that are simultaneously useful for future task solving and future

self-improvement.

After the agent terminates, the runner extracts the modified skills, computes the diff against the original skills, and discards any candidate whose diff is empty. Each non-empty candidate is then validated by re-running the modified agent on the same task in a fresh container; we compute its validation reward gain over the current active agent on that task and select the candidate with the largest strictly positive gain within the batch.

The task’s submitted solution is fixed before its evaluator becomes available, so this validation cannot retroactively improve the reward that generated the modification. It is a safety filter for future branch continuation. A task-specific overfit may pass this filter, but it will not improve the performance metric.

C.4 Complete Online Protocol

Algorithm 1 specifies the execution mechanism. The notation uses one online batch per iteration; when a batch contains multiple tasks, solving, submission, evaluation, and candidate generation are performed per task before the single batch-level tree update.

C.5 Required Computation Resources

Our SWE-Bench Pro experiments use Qwen3.6-35B-A3B, a mixture-of-experts language model with approximately thirty-five billion total parameters and roughly 3 billion active parameters per token, served locally through vLLM on NVIDIA A100 GPUs. Our DeepResearch Bench experiments use Gemini 3 Flash as the task-solving model and Gemini 2.5 Pro as the DeepResearch RACE judge required by the benchmark, both invoked through the Gemini API; the provider does not disclose parameter counts for these models.

In aggregate, all methods and ablations reported in this paper consumed approximately USD 4,000 in API costs and 5,000 A100 GPU-hours. The OASIS runs alone account for approximately USD 1,800 and 2,200 A100 GPU-hours. Because each full stream sweep already consumes a substantial fraction of this budget, we report results from a single run rather than averages over repeated trials. For the same reason, we did not perform a hyperparameter search at the full-run scale, because each end-to-end stream sweep is dominated by API and GPU costs that make repeated runs prohibitive. All reported runs, therefore, share a single fixed configuration.

Algorithm 1 OASIS online update for a task batch

Require: Initial agent π_0 , batches $B_1, \dots, B_T$, levels α_1, α_2

```
1:  $\mathcal{G} \leftarrow (\{\pi_0\}, \emptyset)$ ;  $\pi^{\text{act}} \leftarrow \pi_0$ ;  $\pi^{\text{anc}} \leftarrow \pi_0$ 
2: for  $t = 1, \dots, T$  do
3:    $C \leftarrow \emptyset$   $\triangleright$  validated self-modification candidates
4:   for all  $\tau \in B_t$  do
5:     Run  $\pi^{\text{act}}(\tau)$  and  $\pi^{\text{anc}}(\tau)$  independently
6:     Select one candidate with the LLM judge and submit it
7:     Evaluate both candidates; append rewards and feedback to their histories
8:     Continue the active trajectory to propose  $\tilde{\pi} \leftarrow \text{SELFMODIFY}(\pi^{\text{act}}, \tau)$ 
9:     if  $\tilde{\pi}$  is non-empty then
10:      Add  $\tilde{\pi}$  to  $C$  with its validation gain
11:     end if
12:   end for
13:    $\bar{\pi} \leftarrow \pi^{\text{act}}$ 
14:   if  $C \neq \emptyset$  then
15:     Let  $\bar{\pi}$  be a candidate in  $C$  with largest strictly positive gain
16:     Add  $\bar{\pi}$  to  $\mathcal{G}$  as a child of  $\pi^{\text{act}}$ 
17:   end if
18:    $\pi^+ \leftarrow$  deepest node on the current active lineage satisfying Eq. (7)
19:   if  $\pi^+$  exists then
20:      $\pi^{\text{anc}} \leftarrow \pi^+$ ;  $\pi^{\text{act}} \leftarrow \pi^+$ 
21:   continue
22:   end if
23:    $\pi^- \leftarrow$  earliest non-root node on the provisional path satisfying Eq. (6)
24:   if  $\pi^-$  exists then
25:      $\pi^{\text{act}} \leftarrow \text{parent}(\pi^-)$ 
26:   else
27:      $\pi^{\text{act}} \leftarrow \bar{\pi}$ 
28:   end if
29: end for
```
