SELF-IMPROVEMENT VIA FAST TREE-SEARCH

Xinghong Fu
MIT, Sakana AI

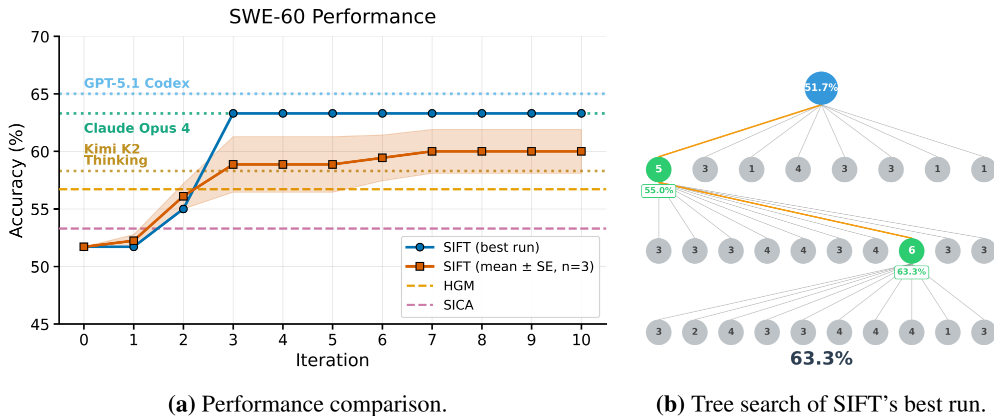
Aravinth Kulanthaivelu
Sakana AI

Yutaro Yamada
Sakana AI

ABSTRACT

Coding agents can recursively modify their own implementations, forming a loop of self improvement. While prior work shows this can boost performance on coding benchmarks, existing approaches are costly and compute-intensive. We introduce a simple, sample-efficient self-improvement framework that significantly improves coding performance under strict budget constraints. We identify evaluation of candidate self-modifications as the main bottleneck since prior approaches estimate their effectiveness by re-running a subset of benchmark tasks with the modified agent. We replace these partial benchmark runs with an LLM-as-a-judge signal to rank candidate patches, reserving expensive evaluations only for the most promising ones. Combined with a lightweight tree search, this enables effective exploration with minimal overhead. On a subset of SWE-bench Verified, our method improves `gpt-5-mini` pass rate by over 11% in fewer than three self-improvement steps, using just $25 in API costs within 15 CPU hours.

1 INTRODUCTION



(a) Performance comparison. **(b)** Tree search of SIFT's best run.

Figure 1: **(a)** Performance on SWE-60 by **SIFT**, averaged across 3 runs, compared against several baselines. SIFT starts with the `mini-swe-agent` harness on `gpt-5-mini` and improves to match the performance of Claude 4 Opus. Each iteration corresponds to the full-evaluation of a solution on SWE-60. **(b)** Tree search leading to the best solution from SIFT. The integers labeling intermediate nodes indicate score assigned by Judge. Note that only the iterations corresponding to accepted solutions, or the last iteration, are shown.

Large Language Models (LLMs) have enabled the rapid development of autonomous agents, capable of reasoning, planning, and executing complex workflows. Among these, coding agents have made leapfrog gains within the past years, demonstrating the ability to solve increasingly difficult software engineering tasks (Jimenez et al., 2024; Jain et al., 2025). One particular frontier within this domain is recursive self-improvement: the ability of an agent to modify its own source code to enhance its future performance.

Theoretical frameworks such as the Gödel Machine (Schmidhuber, 2006) propose that a program capable of provably beneficial self-modifications will converge to a global optimum. Recent empirical implementations, such as the Gödel Agent (Yin et al., 2025), SICA (Robeyns et al., 2025), and the Darwin-Gödel Machine (DGM) (Zhang et al., 2025b), have validated this potential, showing that agents can indeed navigate their own design space to refine their own implementation, with the

improvement validated through competitive results on various benchmarks. However, this search process is extremely inefficient. Current methods rely on an expensive feedback loop where every candidate modification must be validated against a comprehensive benchmark suite to estimate its utility. As noted in recent works, this evolutionary process can incur astronomical costs, upwards of $22,000, and consume hundreds of CPU hours (Zhang et al., 2025b; Wang et al., 2025). This high barrier to entry limits the accessibility of self-improving agents and slows the pace of research.

In this paper, we identify that the primary bottleneck in self-improvement is the low **signal-to-cost ratio** of the evaluation step. Validating a self-improvement patch via full benchmark execution provides a strong signal but at maximum cost. We propose that this process can be optimized by introducing intermediate, cheaper signals via a separate LLM-judge tasked with rating the quality of a self-improvement proposal. We introduce **SIFT** (Self-Improvement via Fast Tree Search), a simple framework that dramatically reduces the time and financial cost of self-improvement while still enabling the discovery of meaningful self-improvement modifications, maintaining competitive performance gains.

Our key contributions include

1. **LLM-as-a-judge for evaluation**. We identify the evaluation of each self-improvement patch to be the most cost-heavy part of the self-improvement process, and show that we can use LLM-as-a-judge to achieve moderately good signal on which patches potentially lead to more performance gains. This allows us to limit our full benchmark evaluation to the most promising nodes and significantly reduce the cost of each self-improvement step.
2. **Sample efficient tree search.** We show that repeated self-improvement sampling guided by a strong LLM-as-a-judge during a simple tree search improves the performance of `gpt-5-mini` on a subset of SWE-bench verified by more than 11% in fewer than 3 self improvement steps, incurring a total API cost of under 25USD within 15 total CPU hours.

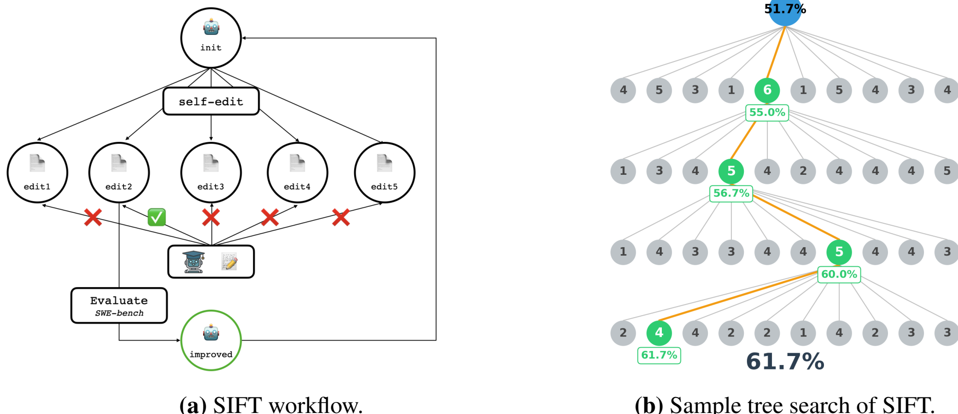
2 Related works

Our work is closely related to an ongoing line of research on self-improving coding agents. The Gödel machine (Schmidhuber, 2006) demonstrates that a program that is able to provably make useful modifications to its own code will find a global optimum solution. The Gödel agent (Yin et al., 2025) proposes sensor-executor modules that implements the approximation of such a program in code, demonstrating improved performance across various language and reasoning benchmarks (Dua et al., 2019; Hendrycks et al., 2021; Rein et al., 2023; Shi et al., 2022). SICA (Robeyns et al., 2025) enforces the self-improvement agent to identically match the agent used during benchmark evaluations, and show significant performance gains in coding (Jimenez et al., 2024; Jain et al., 2025). STOP (Zelikman et al., 2024) uses a meta-utility function to guide a seed-improver to discover multiple self-improvement strategies, such as simulated annealing and genetic algorithm. The Darwin-Gödel Machine (Zhang et al., 2025b) introduces open-endedness through maintaining an archive of well-performing agents to improve the diversity in the agentic design search space. The Huxley-Gödel machine (Wang et al., 2025) proposes the Clade Meta Productivity (Huxley, 1957) metric to sample the next agent based on the performance of a parent agent and all its descendants.

More broadly, this line of work of self-improving coding agents aims to efficiently explore the design space of harness design for agents. ADAS (Hu et al., 2025) demonstrates a Meta Agent Search algorithm that exceeds SOTA performance of human-designed agentic systems across several comprehension and reasoning tasks. AFlow (Zhang et al., 2025c) explores the design space of code-based workflows using Monte Carlo Tree Search, simultaneously achieving performance improvements and cost reductions against stronger LLM backbones. Such work has also been extended to multi-agent systems via MaAS (Zhang et al., 2025a), automating the architecture search by sampling a query-dependent agentic system from a distribution of multi-agent architectures.

Our specific instantiation of agentic design search can be also regarded as a form of meta-learning of coding agents. Within the setting of meta-learning foundation agents, the community has developed methods within prompt evolution (Fernando et al., 2023) in a self-referential manner, and such strategies have recently been shown to outperform reinforcement learning across various tasks (Agrawal et al., 2025). In our work, we allow the agent to evolve not only its prompts through natural language, but also its implementation in code such as tool use error handling.

3 METHODS



(a) SIFT workflow. **(b)** Sample tree search of SIFT.

Figure 2: (a) SIFT workflow. Starting from an inital agent, several self-improvement patches are proposed. They are then scored by a judge and the top-k self-improvement patch is picked for the full evaluation on a coding benchmark. **(b)** Tree search showing recursive self-improvement by SIFT, improving from an inital accuracy of 51.7% to a final accuracy of 61.7% on SWE-60.

Experimental setting We follow the experiment details of HGM (Wang et al., 2025), using `gpt-5-mini` for our coding model, evaluated on the same subset of SWE-bench verified (Jimenez et al., 2024) containing 60 tasks[1]. We refer to this as the SWE-60 subset hereafter.

Our initial harness is based on the `mini-swe-agent`[2], with access to a single `bash` tool that it uses to navigate the `bash-only` environment of SWE-bench. The agent is allowed to make further edits to its codebase freely, in terms of modifications of prompts and additional utility scripts. However, the agent is not allowed to change the environment beyond the `bash-only` setup.

All experiments are conducted in a sandboxed Docker container. During evaluation, we pull the Docker image from the official commit and instantiate our agent in the home directory with `mini-swe-agent` folder together with a simple wrapper script. During self-improvement, our agent is initialized in a standalone Docker container, without any SWE-bench tasks but with access to its own implementation which includes the `mini-swe-agent` subfolder and a single wrapper script. We also pass error logs to this agent.

Self-improvement agent At each iteration of self-improvement, the agent is given the error logs from the previous evaluation on SWE-bench. We note that the full error logs from SWE-bench typically exceed more than 20 turns with a median of 20K tokens per task, sometimes exceeding 50 turns with up to 50K tokens. Hence, passing in the full logs from every failed/incomplete task could exceed context limits or result in heavy context rot. In DGM and HGM, a single random task that failed is sampled and passed as context to a diagnosis agent. However, this results in a failure of the agent to accurately diagnose every point of failure. We propose a simple fix to this problem, which is to also instantiate the diagnosis/self-improvement agent with bash tools to traverse the full directory. In this scaffold, the diagnosis agent typically completes its diagnosis of the problem within 10 turns, traversing up to 10 error directories using `grep, glob, sed` commands while keeping total tokens under 200K. While we note that there are several more advanced methods to handle context rot (Zhang et al., 2026; Anthropic, 2025; Ji, 2025), we find that our basic implementation suffices to handle the current context.

Judge Agent In the evolutionary search for a working self-improvement, the most cost-intensive part is the evaluation. The Gödel machine relies on the strong correlation between performance on coding benchmarks and overall coding agent performance to approximate a “provably" correct signal

[1]The small and medium subsets of SWE-bench Verified used in DGM

[2]https://github.com/SWE-agent/mini-swe-agent

Algorithm 1 Self-Improvement via Fast Tree-search (SIFT)

1: **Input:** Initial agent A_0, scoring judge J, evaluation benchmark $\mathcal{B}$, expansion factor m, iterations N, select top k
2: **Initialize:** Best agent $A^* \leftarrow A_0$, Best score $S^* \leftarrow$ Evaluate$(A_0, \mathcal{B})$
3: **for** $i = 1$ **to** N **do**
4: **Step 1: Agentic Self-Improvement**
5: $\mathcal{P} \leftarrow \emptyset$
6: **for** $j = 1$ **to** m **do**
7: $\mathcal{L} \leftarrow$ Retrieve error logs from previous evaluation of A^* on $\mathcal{B}$
8: $D \leftarrow$ Diagnosis agent performs tool-use traversal of $\mathcal{L}$
9: $P_{new} \leftarrow$ Agent A^* implements patch based on D
10: $\mathcal{P} \leftarrow \mathcal{P} \cup \{P_{new}\}$
11: **end for**
12: **Step 2: LLM-as-a-Judge**
13: $Scores \leftarrow \emptyset$
14: **for** each $P \in \mathcal{P}$ **do**
15: $s \leftarrow J$.score$(A^* + P)$ {Evaluate quality without running $\mathcal{B}$}
16: $Scores \leftarrow Scores \cup \{(P, s)\}$
17: **end for**
18: $\mathcal{P}_{top} \leftarrow$ Top-k patches from $Scores$
19: **Step 3: Full Evaluation**
20: **for** each $P \in \mathcal{P}_{top}$ **do**
21: $A_{new} \leftarrow A^* + P$
22: $S_{new} \leftarrow$ Evaluate$(A_{new}, \mathcal{B})$
23: **if** $S_{new} > S^*$ **then**
24: $A^* \leftarrow A_{new}$
25: $S^* \leftarrow S_{new}$
26: **break** {Greedy acceptance, move to next generation $i + 1$}
27: **end if**
28: **end for**
29: **end for**
30: **Return** Improved agent A^*

Table 1: Resource consumption breakdown per step. Self-improve and LLM-judge costs and durations are reported per task, mean across 10 tasks, using `gpt-5.2`. SWE-60 values are reported as the total value within a 60 task subset of SWE bench, using `gpt-5-mini` and the `mini-swe-agent` harness.

Module	Cost (USD)	Time (CPU Hours)
Self-Improve	0.47	0.067
LLM-judge	0.040	0.0042
SWE-60	3.1	2.9

in self-improvement in order to achieve a globally optimal agent design. However, to obtain a strong signal on the quality of the self-improvement patch requires extensive evaluation on numerous tasks. Even while restricting intermediate evaluations to smaller subsets and using cost-efficient models, the full evolution algorithm results in astronomical costs: 22000USD for SWE-bench using DGM and 5000USD for Polyglot using HGM. Beyond costs, the suite of evaluations also incur significant CPU hours, costing 3 CPU hours for a single evaluation on the SWE-60 which results in hundreds of CPU hours for the full evolution Wang et al. (2025).

We note that a significant hurdle here is the low **signal-to-cost** ratio at each evolution step as it takes tremendous amount of resources to identify good patches from bad ones through downstream task evaluation alone. Instead of running a full downstream evaluation suite, we hypothesize that frontier LLMs are able to distinguish the positive from negative self-improvement patches through critiquing the existing codebase alone (Zheng et al., 2023; Jiang et al., 2025). Importantly, the evaluation process is cheap even when using frontier reasoning models (Table 1), and gives a comparatively strong signal as compared to evaluation on tasks within SWE-bench (Table 2).

We hence propose the use of an LLM-as-a-judge (Zheng et al., 2023) as an intermediate step to rate the proposed patch made by the self-improvement agent according to a rubric from 1 to 10. We then pick the top k patches ranked by the score given and run the full SWE-60 evaluation on these. The best performing patch is then picked as the parent agent for the next iteration of self-improvement.

4 RESULTS

4.1 JUDGE PERFORMANCE

Table 2: Correlation coefficients between the score attained by a single self-improvement patch on SWE-60 and score given by the judge model to the patch. A total of 50 patches are sampled from the same set of models of `gpt-5.2, gpt-5-mini, gpt-5-nano`. Cost is reported per run, averaged across the 50 runs.

Judge	Pearson	Spearman	Cost (USD)
`gpt-5.2`	0.340	0.145	0.040
`gpt-5-mini`	0.286	0.044	0.0057
`gpt-5-nano`	0.076	-0.157	0.0011
Random subtask	0.220	0.138	0.051

We hypothesize that a strong judge is able to provide efficient signals on the improvement potential of a suggested patch. We demonstrate this with the following experiment. Starting from our base model, we sample 50 patches of varying quality. Then, each of these 50 patches is evaluated on SWE-60 and simultaneously scored by a judge. The correlations between the judge score and the SWE-60 score is reported in Table 2. That is, given $N = 50$ total suggested self-improvement patches, we report the correlation for judge model $\mathcal{M}$ with benchmark $\mathcal{B}$ as

$$\mathrm{Corr}(\mathcal{M}, \mathcal{B}) = \mathrm{Corr}_{1 \leq i \leq N}(s_{i,\mathcal{M}}, \mathrm{Acc}_i)$$

where the Spearman rank correlation is computed by ranking both the score $s_{i,\mathcal{M}}$ assigned to patch i by model $\mathcal{M}$ and ranking the accuracy obtained on SWE-60 Acc_i by patch i.

We observe that judge scores have roughly a 34% correlation with the benchmark performance, with correlation increasing with model strength. This corroborates with existing studies (Jiang et al., 2025) which show that reasoning models significantly outperform non-reasoning variants as a code judge, providing a cost-efficient method for obtaining a signal to distinguish between good and bad self-improvements as `gpt-5.2(high reasoning)` cost less than 0.05 USD per task.

For a baseline comparison, we compare this with the correlation between the success of a random task on SWE-60 and the overall accuracy obtained on SWE-60. Explicitly, across each task t within the $T = 60$ tasks, we compute

$$\mathbb{E}_{t \in T}[\mathrm{Corr}_{1 \leq i \leq N}(\mathbb{1}_{t,i}, \mathrm{Acc}_i)]$$

where $\mathbb{1}_{t,i}$ is the indicator random variable for whether task t was solved on patch i. This metric is reported in the last row of Table 2. At roughly the same cost, using a frontier LLM judge gives a stronger signal on the quality of the self-improvement over evaluating on a random task within the subset. Also note that the API calls to the LLM-judge is much faster than the time required than a benchmark task evaluation (Table 1), which makes the LLM-judge both more cost and compute efficient than downstream task evaluation.

These scoring signals from the judge can also be used in multiple rounds of self-improvement. Figure 3 shows the binned accuracy improvement due to a self-improvement patch plotted against the score assigned by a `gpt-5.2` judge to the patch. Majority of proposed patches achieve rather mediocre scores. At later iterations of self-improvement, low-scoring patches also can sometimes result in severe degradation of performance by introducing more bugs. Nonetheless, higher scoring patches tend to deliver strong performance gains.

4.2 SWE-VERIFIED-60

To verify the extent of self-improvement achieved, one can evaluate the improved model on a task correlated to the performance of a model. In terms of coding agents, this is best realized through

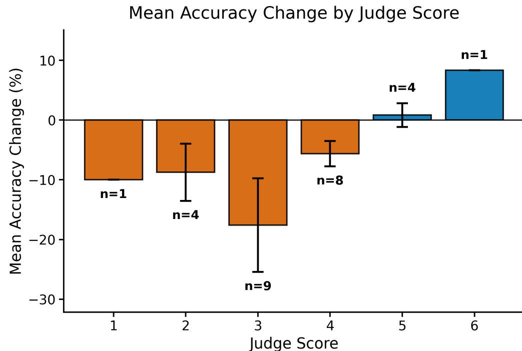


Figure 3: Mean accuracy change against score by judge LLM(`gpt-5.2`), averaged across the three runs in Figure 1. Error bars represent standard errors.

a coding benchmark. Tasks within SWE-bench-verified Jimenez et al. (2024) includes PRs from existing Github repositories requiring the implementation of a feature or bug fixes that were previously tasked to humans. These tasks help to verify the general capabilities of coding agents, such as code understanding, multi-file handling and long context management, ideal attributes exhibited by a coding agent. Existing works (Wang et al., 2025; Zhang et al., 2025b) have also found self-improving coding agents to show improved performance on this benchmark over multiple rounds of self-improvement.

In our experiments, we use the same subset of SWE-bench-verified as in Wang et al. (2025) due to the extensive cost demanded by the full benchmark. At each iteration, we sample 10 self-improvement patches and select the top patch ranked by the score given by the judge LLM. The top scoring patch is then evaluated on the SWE-60 tasks and accepted if the accuracy improves. Otherwise, the patch is rejected and 10 new patches are resampled independently on the next iteration. Hence, each iteration on Figure 1 corresponds to exactly one full evaluation on SWE-60.

Within the first 10 iterations, we consistently observe SIFT to achieve an improvement from the initial 51.7% to a 60%(Figure 1). This exceeds the accuracy of Kimi-K2-Thinking(which is the current best open-source model) and matches the same level of performance as Claude Opus 4 using the `mini-swe-agent` harness, demonstrating that a weaker model can rival the performance of stronger models when bootstrapped with a stronger harness.

The performance of SIFT on SWE-60 also exceeds that of existing self-improvement frameworks, such as HGM and SICA at lower cost and CPU hours. However, we note that this comparison is not entirely fair as the initial benchmark scores of HGM and SICA were 40% while that for SIFT was 51.7%. Nonetheless, our results still demonstrate the viability of SIFT as a fast and effective framework to achieve self-improvement in coding abilities.

Ablation We ablate SIFT against a simpler baseline to verify its effectiveness. We compare SIFT against `single-sample`, where just a single proposed patch is sampled at each step of self-improvement. The judge necessarily only scores and selects a single patch. This is reflected as `single-sample` in Figure 4. Note that this is equivalent to replacing the judge with a random sampler.

We observe that SIFT consistently outperforms `single-sample`, averaged across three separate runs at little compute and cost overhead, highlighting the added benefit of a judge LLM.

4.3 POLYGLOT

To verify the generalizability of the SIFT method to other coding tasks, we run another set of experiments on Polyglot, following Wang et al. (2025) and Zhang et al. (2025b). Polyglot (Gauthier, 2024) is a multilingual coding benchmark where the agent is tasked to implement solutions to word problems across the languages `c++, go, rust, java, javascript, python`. Due to the larger number of tasks on Polyglot, we report mainly exploratory results on the effectiveness of SIFT to achieve few-step self-improvement.

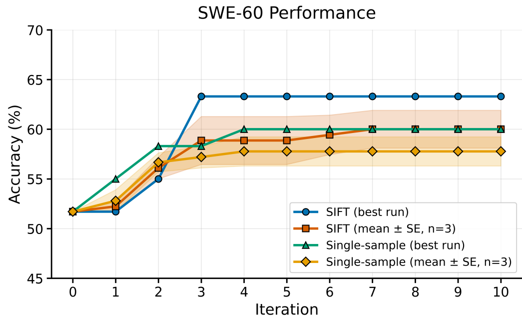


Figure 4: Comparison of SIFT against a simpler baseline where we just sample a single modification patch without a judge LLM at each step of self-improvement.

We begin with a similar harness as implemented in previous baselines Wang et al. (2025); Zhang et al. (2025b) and obtain an initial starting accuracy of 22.7% on Polyglot. Through a single step of self-improvement, the top scoring patch received a score of 8 from the judge LLM and the improved agent achieves a performance gain of up to 27.7%, nearing the performance of the best existing self-improvement frameworks on Polyglot.

Strong correlation is also observed between scores assigned by the LLM-judge to the extent of self-improvement(Figure 5), corroborating our findings on SWE-60 that a strong judge can act as a signal for the effectiveness of self-improvement at a fraction of the cost of benchmark evaluation. Across 5 independent patches, we find a stronger signal from the LLM-judge as compared to randomly sampling a task from Polyglot. We hope to expand these experiments to arrive at a stronger conclusion.

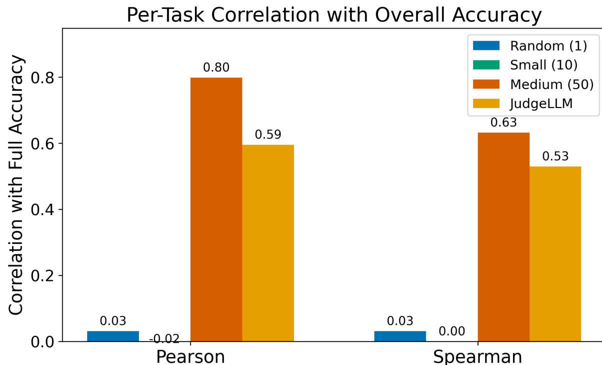


Figure 5: Correlation between the score provided by the judge LLM(`gpt-5.2`) and the full 225 tasks on Polyglot. For reference, we also compute the average(across all 225 tasks) of correlation of the task success rate with overall benchmark accuracy, as well as the small(10 task) and medium(50 task) subset performance correlation with the full Polyglot benchmark.

The high correlation of the LLM judge score and the Polyglot performance allows for multiple iterations of the greedy tree search algorithm following SIFT 1. We set 10 parallel self-improvement agents, and a single judge, each using `gpt-5.2` as the LLM backbone. We select the top scoring patch(scored by the judge LLM) at each step and use this as the agent for the next step of self-improvement. By running SIFT for two iterations, we improve the benchmark score on Polyglot from 22.7% to 28.4%.

5 Discussion and Conclusion

In this paper, we introduced **SIFT**: Self-Improvement via Fast Tree-search. We propose to replace the downstream task evaluation, cost and compute-intensive process of recursive self-improvement, with a judge LLM at each intermediate step. We observe that models with stronger reasoning abilities are significantly more capable than weaker models at distinguishing the quality of proposed

self-improvement patches. Using a strong reasoning model for the LLM-judge, we significantly speed up the process of self-improvement, achieving a performance gain exceeding 10 percentage points on a 60 task subset of SWE-bench within 3 evaluation steps, and achieving 6% performance gain to 28.4% on Polyglot in under 2 self-improvement steps.

Limitations The main drawback of this work is the limited evaluation subset used. Our experiments were performed on a subset of SWE-bench that, while matching the subset used in related works (Wang et al., 2025; Zhang et al., 2025b), still only contained tasks from `django` and `sphinx`. While our experiments on Polyglot were conducted on the full benchmark, we still hope to expand these experiments more thoroughly in the future. In spite of the difficulty in correlating the performance of a coding agent to any single benchmark due to potential overfitting, extensive work has been done in expanding the world of benchmarking tasks (Chen et al., 2026) and we hope that SIFT can be an efficient framework in exploring the design space of coding agent with the help of these more extensive benchmarks.

We also note that the initial harness used in our implementation exceeds the baseline performance of that in other works. However, we believe that the implementation of our current harness most closely matches the performance at the frontier of coding agents[3]. A significant gain starting from a weak harness that does not improve upon the state-of-the-art yields little contribution as compared to one that improves upon a stronger harness closer to frontier performance. As adequately noted in Wang et al. (2025), a stronger harness is more difficult to improve from. Our experiments reveal similar trends, as we observe lower judge scores and smaller improvements as the accuracy on a benchmark saturates. Nonetheless, SIFT consistently improves from its baseline harness efficiently across different benchmarks.

SIFT is also not entirely *self*-improving in the strictest sense because it relies on an external judge LLM to assign scores to self-improvement patches. While pre-existing works that we build on also make distinctions between the diagnosis model and the coding model (Wang et al., 2025; Zhang et al., 2025b), we believe that using the same model and harness for each part of the pipeline most accurately demonstrates self-improvement abilities, which we aim to include into our work in the future.

Future work SIFT implements a greedy tree search algorithm, where the parent node for the next iteration of self-improvement is always selected to be the best performing node from the previous generations. However, existing literature suggests that evolutionary pressure from the competition across different generations can lead to a stronger agent discovered in the long run. SIFT has demonstrated LLM-as-a-judge has strong evaluation capabilities across a single iteration of self-improvement. The use of a judge LLM for evaluation across multiple generations of the tree archive can potentially lead to more interesting discoveries.

Existing literature has also explored the use of an agent as a judge (Zhuge et al., 2024) to evaluate agents with agents. Such a framework can also be implemented within SIFT, by turning the judge LLM into a judge agent. This would entail giving abilities to the judge LLM to selectively run coding benchmark tests against suggest patches. Although this increases compute cost at the judging stage, it may allow the judge to form a more accurate estimate of a patch's self-improvement potential.

Safety and Ethics This work aims to advance the field of machine learning and artificial intelligence and we acknowledge that several risks are correlated with the increased capabilities of AI agents through recursive self-improvement. Several examples include unsandboxing and environment hacking. We observe that when unrestricted, the diagnosis agent can suggest patches which modify the time limit, or increase the number of retry attempts in the environment. These were immediately fixed through a direct prompt to the agent and rejecting patches to unauthorized files, and none of these behaviors made it through to the final experiments. Nonetheless, we believe that safety remains a concern for more complex agentic systems and should be handled with care when conducting research in recursive self-improvement.

[3] verified against the official SWE-bench leaderboard https://www.swebench.com/

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2025. URL https://arxiv.org/abs/2507.19457.

Anthropic. Effective context engineering for AI agents. Anthropic Engineering Blog, 2025. URL https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents. Accessed: February 7, 2026.

Mouxiang Chen, Lei Zhang, Yunlong Feng, Xuwu Wang, Wenting Zhao, Ruisheng Cao, Jiaxi Yang, Jiawei Chen, Mingze Li, Zeyao Ma, Hao Ge, Zongmeng Zhang, Zeyu Cui, Dayiheng Liu, Jingren Zhou, Jianling Sun, Junyang Lin, and Binyuan Hui. Swe-universe: Scale real-world verifiable environments to millions, 2026. URL https://arxiv.org/abs/2602.02361.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs, 2019. URL https://arxiv.org/abs/1903.00161.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution, 2023. URL https://arxiv.org/abs/2309.16797.

Paul Gauthier. o1 tops aider's new polyglot leaderboard. https://aider.chat/2024/12/21/polyglot.html, December 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems, 2025. URL https://arxiv.org/abs/2408.08435.

Julian Sorell Huxley. The three types of evolutionary process. *Nature*, 180:454–455, 1957. URL https://api.semanticscholar.org/CorpusID:4174182.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Yichao Peak Ji. Context engineering for AI agents: Lessons from building Manus. Manus Blog, July 2025. URL https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus. Accessed: February 7, 2026.

Hongchao Jiang, Yiming Chen, Yushi Cao, Hung yi Lee, and Robby T. Tan. Codejudgebench: Benchmarking llm-as-a-judge for coding tasks, 2025. URL https://arxiv.org/abs/2507.10535.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

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

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. In *Scaling Self-Improving Foundation Models without Human Supervision*, 2025. URL https://openreview.net/forum?id=rShJCyLsOr.

Juergen Schmidhuber. Goedel machines: Self-referential universal problem solvers making provably optimal self-improvements, 2006. URL https://arxiv.org/abs/cs/0309048.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. Language models are multilingual chain-of-thought reasoners, 2022. URL https://arxiv.org/abs/2210.03057.

Wenyi Wang, Piotr Piękos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-gödel machine: Human-level coding agent development by an approximation of the optimal self-improving machine, 2025. URL https://arxiv.org/abs/2510.21614.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1354. URL https://aclanthology.org/2025.acl-long.1354/.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation, 2024. URL https://openreview.net/forum?id=1gkePTsAWf.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models, 2026. URL https://arxiv.org/abs/2512.24601.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet, 2025a. URL https://arxiv.org/abs/2502.04180.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents, 2025b. URL https://arxiv.org/abs/2505.22954.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025c. URL https://openreview.net/forum?id=z5uVAKwmjf.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents, 2024. URL https://arxiv.org/abs/2410.10934.