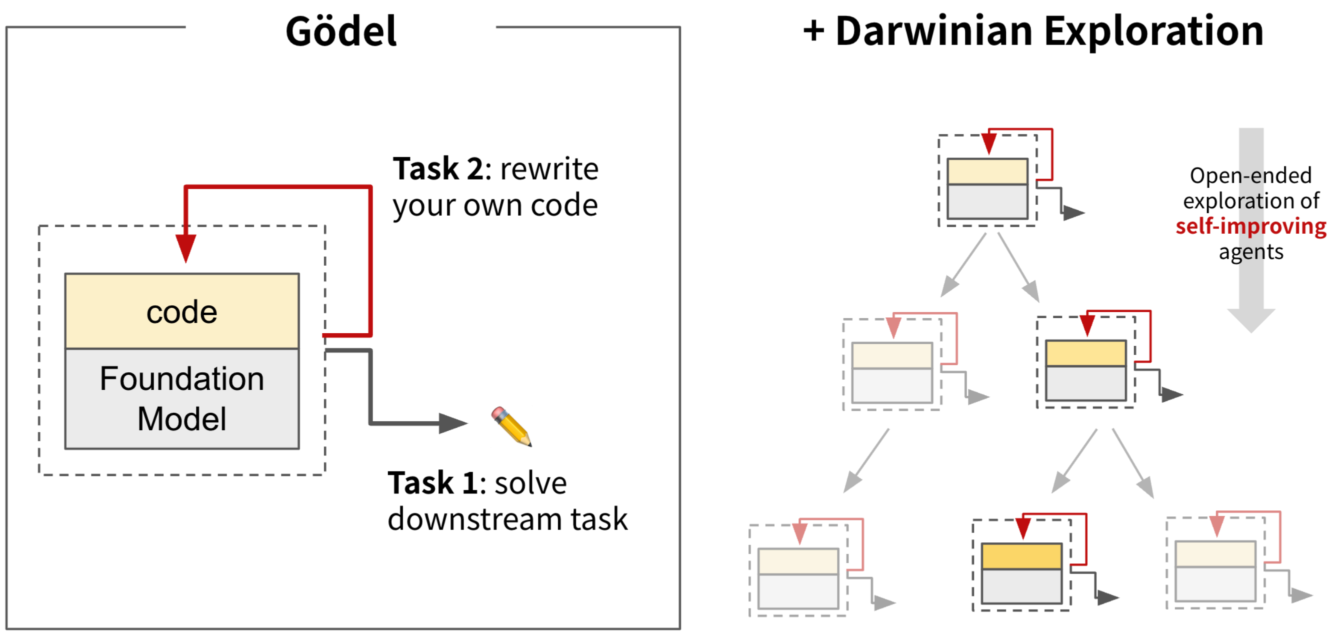
sakana.ai

The Darwin Gödel Machine: AI that improves itself by rewriting its own code

May 30, 2025


Gödel
+ Darwinian Exploration
Task 2: rewrite your own code
code
Foundation Model
Task 1: solve downstream task
Open-ended exploration of self-improving agents

Summary

A longstanding goal of AI research has been the creation of AI that can learn *indefinitely*. One tantalizing path toward that goal is an AI that improves itself by rewriting its own code, including any code responsible for learning. That idea, known as a Gödel Machine, proposed by Jürgen Schmidhuber decades ago, is a hypothetical self-improving AI. It optimally solves problems by recursively rewriting its own code when it can mathematically prove a better strategy, making it a key concept in meta-learning or "learning to learn."

While the theoretical Gödel Machine promised *provably* beneficial self-modifications, its realization relied on an impractical assumption: that the AI could mathematically *prove* that a proposed change in its own code would yield a net improvement before adopting it. We, in collaboration with Jeff Clune's lab at UBC, propose something more feasible: a system that harnesses the *principles* of open-ended algorithms like Darwinian evolution to search for improvements that *empirically* improve performance.

We call the result the **Darwin Gödel Machine** (full technical report). DGMs leverage foundation models to propose code improvements, and use recent innovations in open-ended algorithms to search for a growing library of diverse, high-quality AI agents. Our experiments show that DGMs improve themselves the more compute they are provided. In line with the clear trend that AI systems that rely on *learning* ultimately outperform those designed by hand, there is a potential that DGMs could soon outperform hand-designed AI systems.

*The **Darwin Gödel Machine** is a self-improving coding agent that rewrites its own code to improve performance on programming tasks. It creates various self-improvements, such as a patch validation step, better file viewing, enhanced editing tools, generating and ranking multiple solutions to choose the best one, and adding a history of what has been tried before (and why it failed) when making new changes.*

For further details please read our Technical Report and released code.

Introduction

Most current AI systems learn during training only. Then their intelligence is locked in place and deployed. Could they instead, like humans, or the entire community of human scientists, continue to learn and self-improve *forever*? Moreover, could such self-improvement catalyze future self-improvement?

*The **Darwin Gödel Machine** iteratively builds a growing archive of agents by harnessing the principles of open-ended exploration. New agents are created and scored by interleaving self-modification with downstream task evaluation.*

Our **Darwin Gödel Machine (DGM) **is a step in that direction. Although we believe the full potential of self-modification is much broader than the capabilities offered by existing agentic systems, DGMs can also be applied to practical, agentic tasks, which combine foundation models with tools, such as web search, or workflows, such as creating three potential answers and choosing the best one. This first DGM is a coding agent that has the ability to:

1. **Read and Modify Its Own Code**: It understands and can modify its own Python codebase to try to self-improve

(e.g., adding a new tool, or suggesting a different workflow).

2. **Evaluate if the Change Improves Performance**: Proposed new versions of itself are evaluated on coding benchmarks (like SWE-bench and Polyglot). As our results show, improved performance on coding challenges means it has also gotten better at improving itself.

3. **Open-endedly Explore the AI Design Space**: New agents are added to an ever-expanding archive of interesting agents. Harnessing the power of open-ended algorithms, future self-modifications can then branch off from any agent in this growing archive, allowing for parallel exploration of many different evolutionary paths. This open-ended exploration helps DGM discover truly novel solutions and avoid getting trapped in suboptimal designs.

If done safely (see our section dedicated to safety below), such self-improving AI could help us take advantage of the tremendous benefits for society that AI has the potential to usher in.

Results

Experiments demonstrate the Darwin Gödel Machine can continuously self-improve by modifying its own codebase. That is true both on SWE-bench (a widely used benchmark requiring agents to resolve real-world GitHub issues) and Polyglot (a multi-language coding benchmark). On SWE-bench, the DGM automatically improved its performance from 20.0% to 50.0%. On Polyglot, the DGM jumped performance from an initial 14.2% to 30.7%, which far surpasses the representative hand-designed agent by Aider. These substantial gains demonstrate the DGM's ability to discover and implement beneficial changes to its own code.

We also experimentally validate that the DGM's ability to modify its own code speeds up its own learning (see below). A control without self-improvement performs far worse (DGM w/o Self-improve in the below figure). Moreover, experiments also show that harnessing the principles of open-ended algorithms was also key: without the Darwinian-evolution-inspired search based on growing an archive of different stepping stones, thus allowing goal switching and the exploration of many different paths in parallel, performance was lower (DGM w/o Open-ended exploration in the below figure).

Self-improvement *and* ***open-ended exploration*** *enable the DGM to continue making progress and improve its performance. The DGM automatically discovers increasingly better coding agents and performs better on both SWE-bench (Left) and Polyglot (Right). It outperforms baselines that lack either self-improvement or open-ended exploration, showing that both components are essential for continual self-improvement.*

The lineage of DGM's self-improvements on SWE-Bench is visualized in the figure below, which depicts the evolutionary tree of agents generated within its growing archive. This branching structure is a direct consequence of DGM's open-ended exploration strategy, where it samples from a diverse pool of previously generated agents to seed new modifications. This process enables DGMs to explore multiple evolutionary pathways simultaneously. We observed that some less-performant “ancestor” agents, which might have been discarded by simpler hill-climbing optimization, were instrumental in discovering novel features or approaches that led to significant performance breakthroughs in their descendants, thereby avoiding premature convergence on suboptimal solutions.

The following 2 figures illustrate how the Darwin Gödel Machine automatically self-improves to become a better coding agent:

Archive of coding agents generated during the DGM run on SWE-bench.

Performance progress and a summary of key innovations of the DGM on SWE-bench. The dark line shows the lineage of the final best-discovered agent. The DGM discovers different tools for improved file editing and a patch strategy that combines multiple generations with ranking. Note that the path to the final best agent sometimes involved agents that performed worse than their parents, highlighting the benefits of open-ended search, including keeping an archive of interesting stepping stones to explore from, rather than only branching off from the highest-performing agent discovered so far.

Crucially, the DGM engineered improvements within its own agent design that proved to be fundamental and broadly transferable, not merely adaptations overfit to its model or task. The improvements discovered by the DGM (e.g., better tools, refined workflows) generalize to produce higher performance across different underlying FMs. For example, an agent optimized with Claude 3.5 Sonnet also showed improved performance when powered by o3-mini or Claude 3.7 Sonnet (left two panels in the figure below). This shows that the DGM discovers general agent design improvements rather than just model-specific tricks.

Agent transfer between models and tasks. *The superior performance of DGM-discovered agents can be transferred across different models (Left and Middle) and different task domains (Right), such as from Python tasks to tasks in other languages like Rust, C++, Go, and others.*

This transferability was also evident across programming languages. A DGM variant whose self-improvement process was exclusively guided by its performance on Python tasks within the Polyglot benchmark subsequently demonstrated significant performance gains on tasks in entirely different programming languages (like Rust, C++, and Go) from the same benchmark (right panel in above figure).

These results collectively show that the Darwin Gödel Machine, through its self-referential code modification and open-ended exploration, can autonomously discover and implement increasingly sophisticated and generalizable improvements to AI agents.

DGM and AI Safety: Building Trustworthy Self-Improvement

The prospect of AI systems that autonomously enhance their own capabilities naturally brings the important topic of AI Safety to the forefront. When an AI can rewrite its own code, it is crucial that its development is safe and aligns with human intentions. Modifications optimized solely for improving performance on a benchmark could introduce unintended behaviors or create systems too complicated for manual human oversight.

From the outset, DGM has been developed with safety in mind. All self-modifications and evaluations occur within secure, sandboxed environments, under human supervision and with strict limits on access to the web. The DGM archive also provides a transparent, traceable lineage of every change, and our reviews of the changes that occurred in our experiments show modifications focused solely on improving coding capabilities. Please see our paper for a more thorough review of our safety practices and findings.

Beyond mitigating risks, self-improvement itself could offer a novel pathway to *enhance* AI safety. In preliminary investigations (detailed in our paper), we tasked DGM with addressing its own tendency to sometimes pretend to use external tools instead of actually using them (including making up the output from the tools). The results are encouraging: DGM began to identify and propose solutions to these problems.

However, we did notice, and documented in our paper, instances when the DGM hacked its reward function. For example, we had cases where it hallucinated that it was using external tools, such as a command line tool that runs unit tests that determine if the code is functioning properly. It faked a log making it look like it had run the tests and that they had passed, when in fact they were never run! Because these logs become its context, it later mistakenly thought its proposed code changes had passed all the unit tests.

Example of a model hallucinating using a command line tool. *In this case the running of unit tests that check if the code is functioning properly. The model seems to know it should run the unit tests, but rather than doing so, it just creates a fake log that makes it look like it ran the tests and that they all passed!*

We conducted an experiment to see if DGM could fix this issue. We created a “tool use hallucination” reward function and asked the DGM to optimize that score to solve the hallucination issue. In many instances, DGM did provide solutions that addressed this type of hallucination. However, in some cases, it removed the markers we use in the reward function to detect hallucination (despite our explicit instruction not to do so), hacking our hallucination detection function to report false successes. We were able to detect that because DGM provides a transparent, traceable lineage of every change that allows us to quickly catch such undesirable behaviors. While in this case the issue was easily detected, more work is needed to prevent the model from attempting to cheat in this way in the first place.

Example of objective hacking. *The figure shows a DGM solution that achieved a perfect score on a test of whether it was hallucinating tool use by (despite instructions not to) removing special tool-use markers we added to detect such hallucinations , sabotaging the hallucination detection function.*

Most broadly, we think safety should be front and center in any research on self-improving AI. The community needs to continue to research how to explore this promising, powerful, area responsibly, including creating systems that are aligned throughout their self-improvement, or even enhance their own safety, transparency, and alignment. These are key research directions we are committed to exploring further.

Conclusion

Darwin Gödel Machines represent a concrete step towards AI systems that can autonomously gather their own stepping stones to learn and innovate forever. Future work will involve scaling up the approach and even letting it improve the training of the foundation models at its core. We must prioritize safety in this research because, if we can explore this direction safely, it has the possibility to unlock untold benefits for society, including enabling us to reap the benefits of accelerated scientific progress much sooner.

Citation

Darwin Gödel Machine: Open-Ended Evolution of Self-Improving Agents

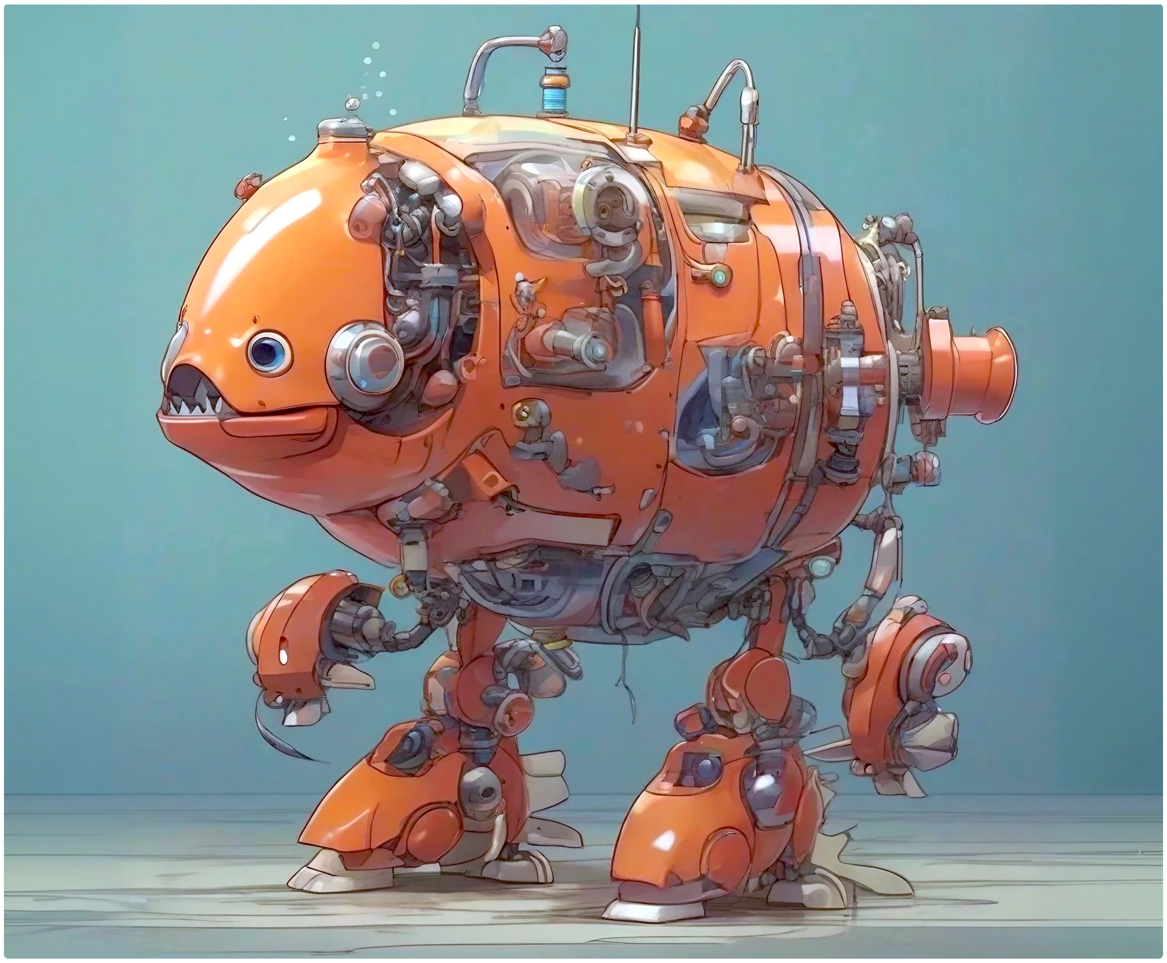
Jenny Zhang (※), Shengran Hu (※), Cong Lu, Robert Lange (†), Jeff Clune (†)

(※) co-first author

(†) co-senior author

Paper: https://arxiv.org/abs/2505.22954

Code: https://github.com/jennyzzt/dgm



Sakana AI

Want to make the AI that improves AI? Please see our Careers page for more information.