

Evolutionary LLM Systems for Code Optimization

Dmytro Nikolaiev, ML Scientist @ ChainML
Jan 2026

Agenda

- Core algorithm & key papers
- Hands-on: OpenEvolve
- Q&A

Main Idea

Given structured feedback, an LLM
can propose code changes that
produce measurable improvements

Main Idea

Given structured feedback, an LLM
can propose code changes that
produce **measurable** improvements

AlphaEvolve by Google DeepMind

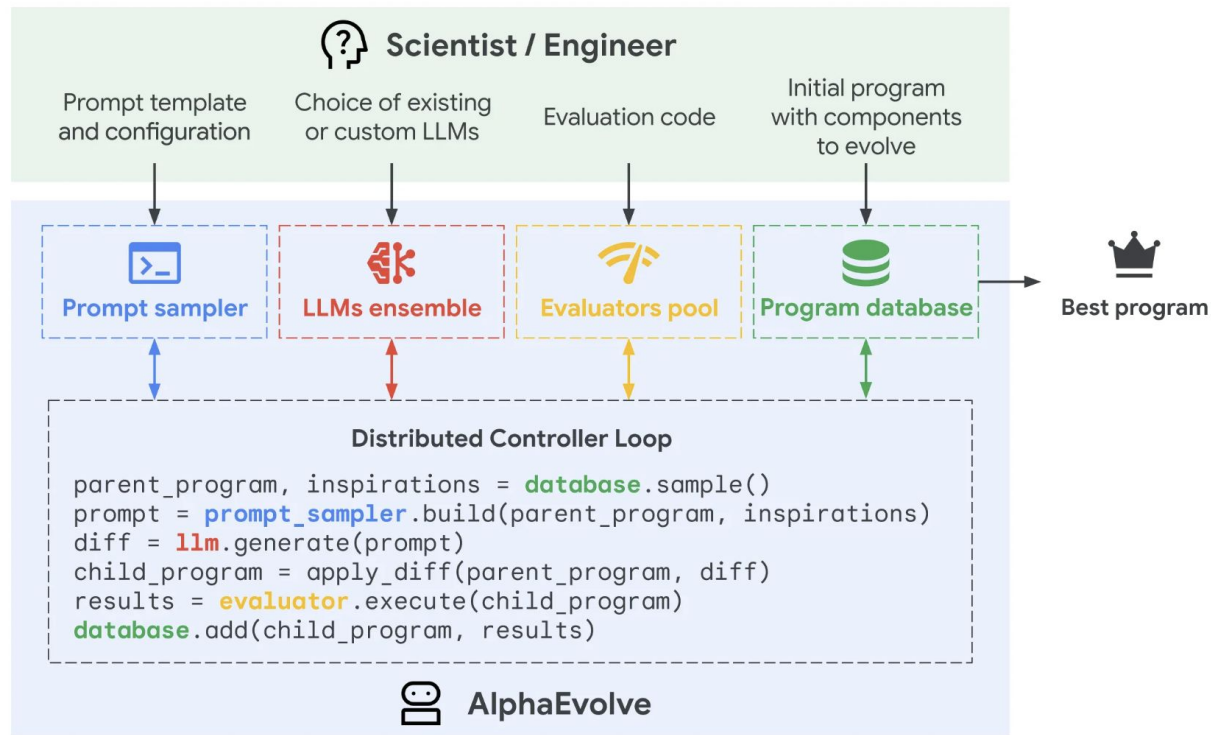


Diagram showing how the prompt sampler first assembles a prompt for the language models, which then generate new programs. These programs are evaluated by evaluators and stored in the programs database. This database implements an evolutionary algorithm that determines which programs will be used for future prompts.

AlphaEvolve by Google DeepMind

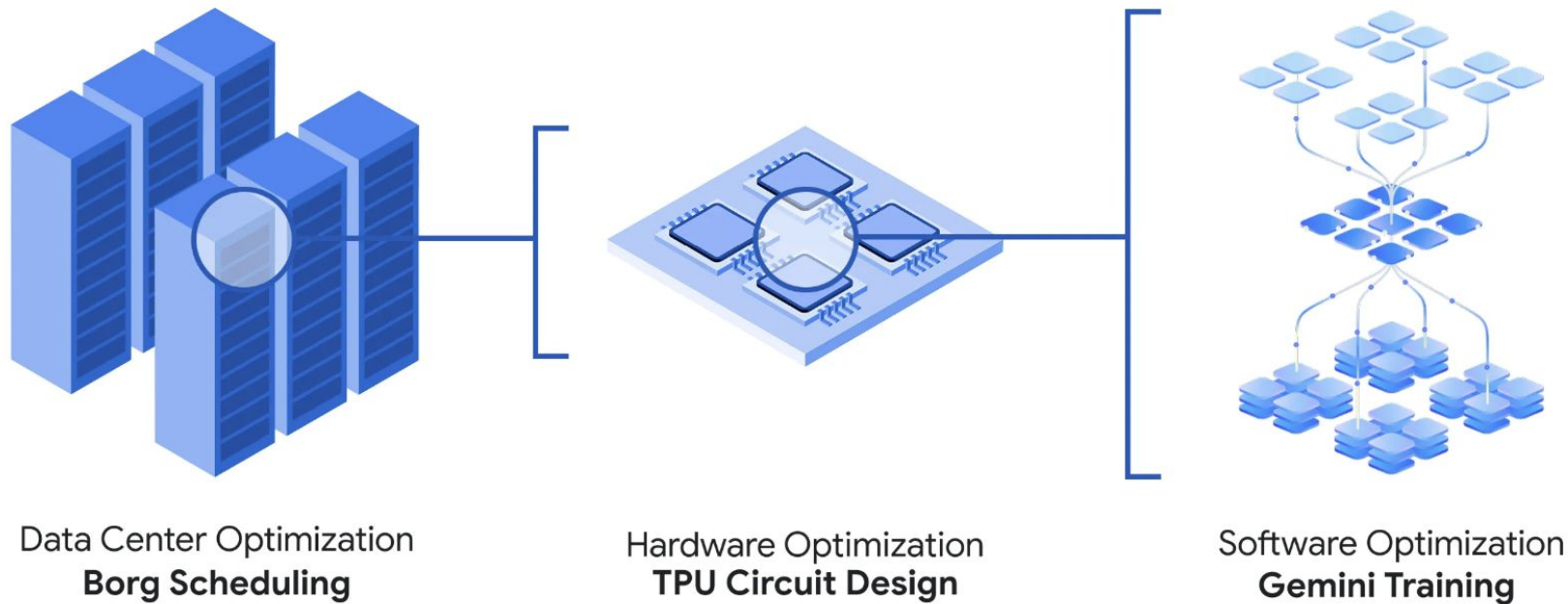


Diagram showing how AlphaEvolve helps Google deliver a more efficient digital ecosystem, from data center scheduling and hardware design to AI model training.

AlphaEvolve by Google DeepMind

Algorithm: AlphaEvolve-like (high-level)

Input: n_{steps}

Initialize: $\mathcal{P} \leftarrow \text{InitializePopulation}()$

for $t = 1, 2, \dots, n_{\text{steps}}$ **do**

$p \leftarrow \text{Select}(\mathcal{P})$

$\hat{p} \leftarrow \text{Mutate}(p)$

$\mathcal{P} \leftarrow \mathcal{P} \cup \text{Filter}(\hat{p})$

end for

AlphaEvolve by Google DeepMind

Algorithm: AlphaEvolve-like (high-level)

Input: n_{steps}

Initialize: $\mathcal{P} \leftarrow \text{InitializePopulation}()$

for $t = 1, 2, \dots, n_{\text{steps}}$ **do**

$p \leftarrow \text{Select}(\mathcal{P})$

p can be a single program or a batch

$\hat{p} \leftarrow \text{Mutate}(p)$

$\mathcal{P} \leftarrow \mathcal{P} \cup \text{Filter}(\hat{p})$

Selection is typically proportional to performance with some exploration to preserve diversity

end for

AlphaEvolve by Google DeepMind

Algorithm: AlphaEvolve-like (high-level)

Input: n_{steps}

Initialize: $\mathcal{P} \leftarrow \text{InitializePopulation}()$

for $t = 1, 2, \dots, n_{\text{steps}}$ **do**

$p \leftarrow \text{Select}(\mathcal{P})$

$\hat{p} \leftarrow \text{Mutate}(p)$

$\mathcal{P} \leftarrow \mathcal{P} \cup \text{Filter}(\hat{p})$

end for

Mutation is usually one or several LLM calls:

- one-shot,
- idea-implementation,
- critique-rewrite,
- agent, etc.

AlphaEvolve by Google DeepMind

Algorithm: AlphaEvolve-like (high-level)

Input: n_{steps}

Initialize: $\mathcal{P} \leftarrow \text{InitializePopulation}()$

for $t = 1, 2, \dots, n_{\text{steps}}$ **do**

$p \leftarrow \text{Select}(\mathcal{P})$

Filtering includes evaluation

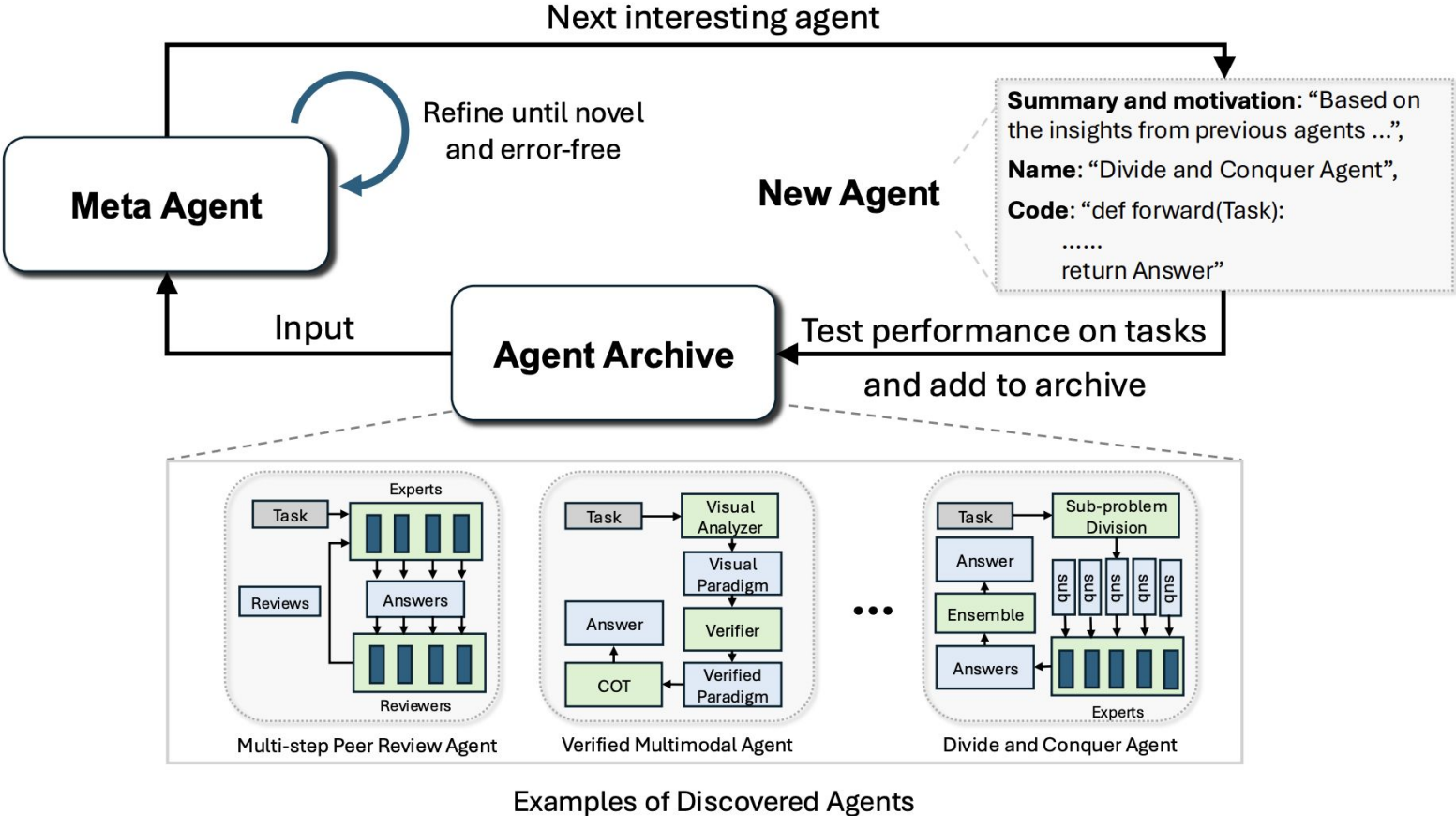
$\hat{p} \leftarrow \text{Mutate}(p)$

$\mathcal{P} \leftarrow \mathcal{P} \cup \text{Filter}(\hat{p})$

Avoid harsh filtering to preserve diversity, e.g. not compiled candidates that have promising ideas

end for

Automated Design of Agentic Systems



Darwin Godel Machine by Sakana AI

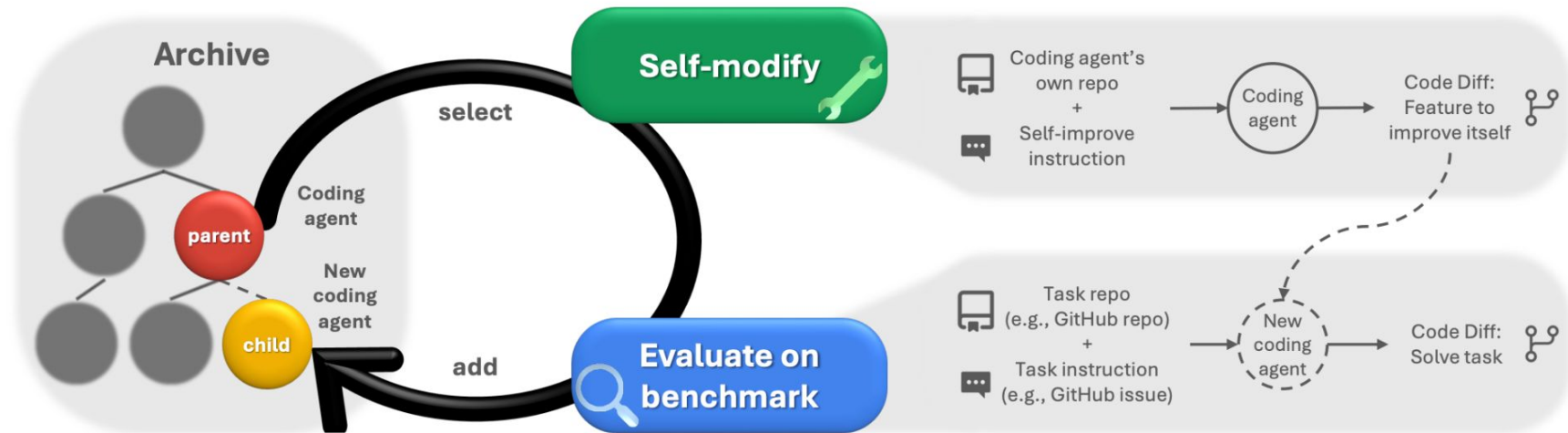
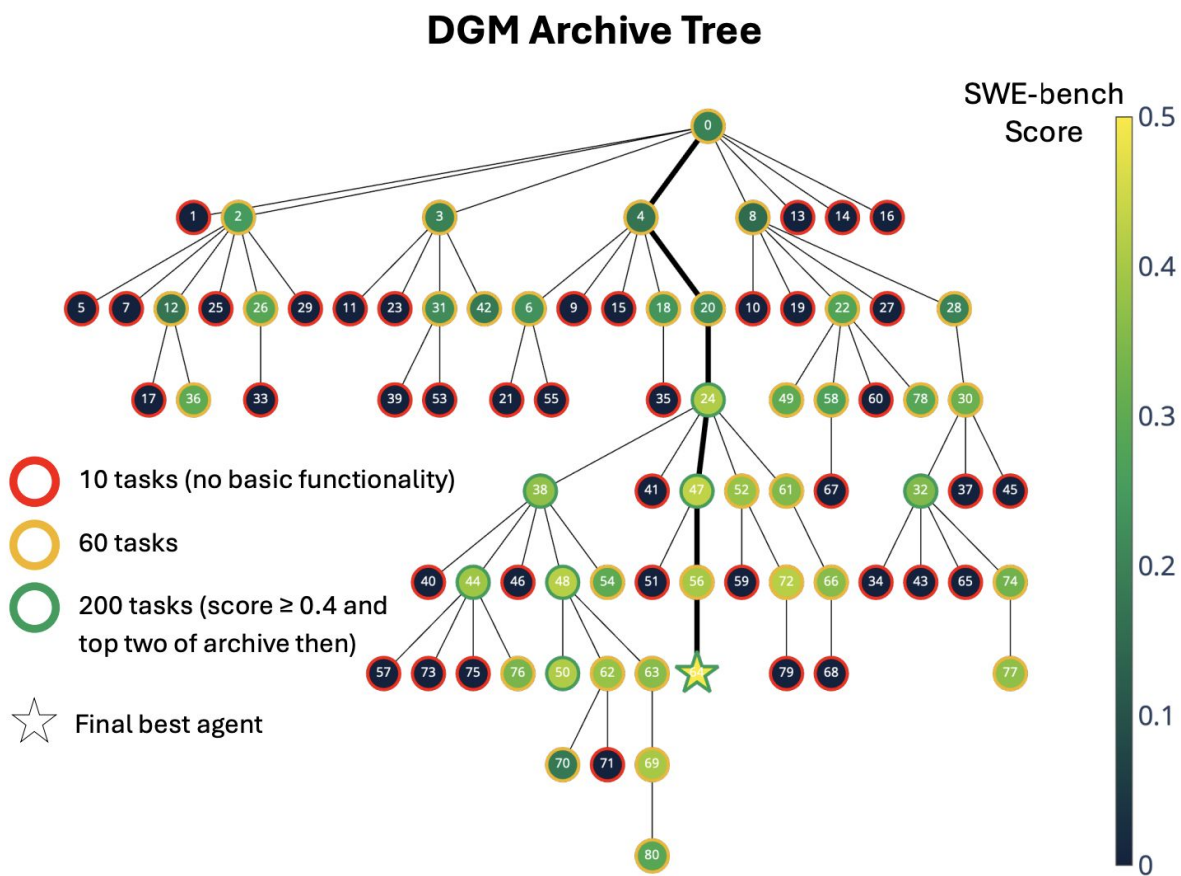
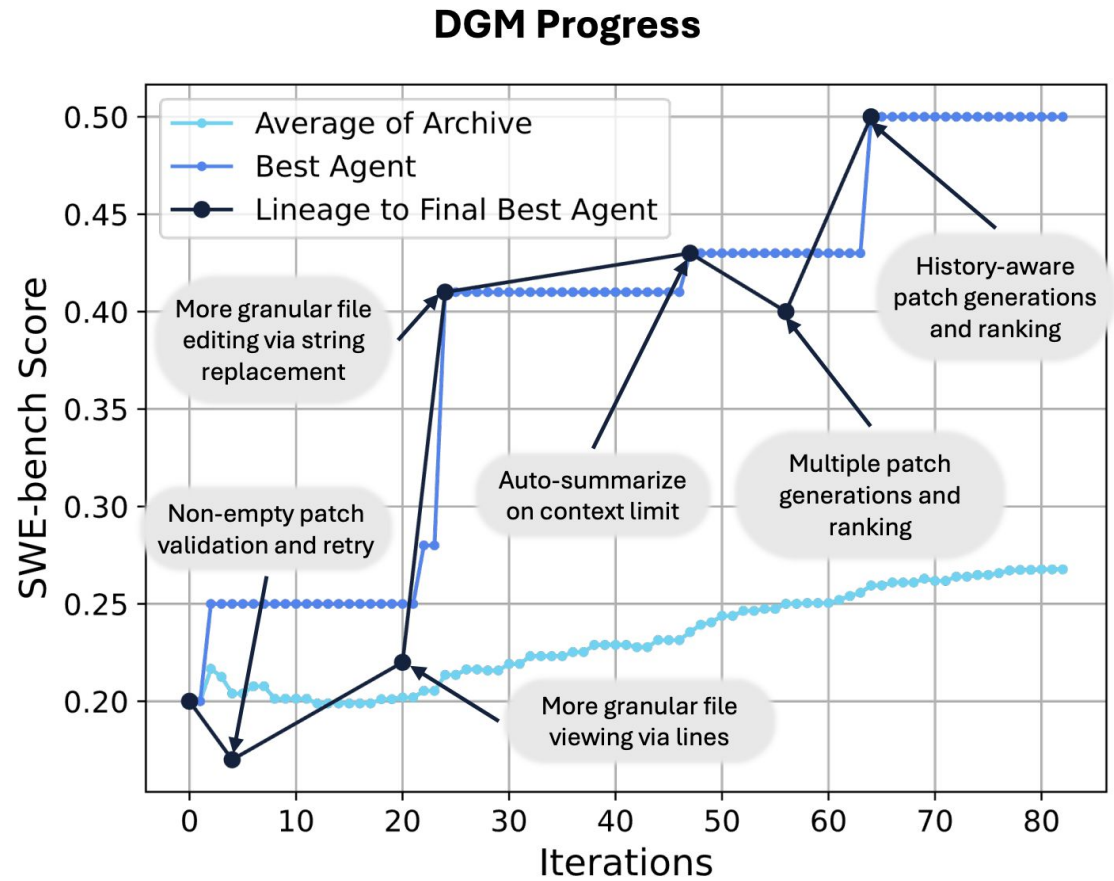


Figure 1: **Darwin Gödel Machine.** The DGM iteratively builds a growing archive of agents by interleaving self-modification with downstream task evaluation. Agents in the archive are selected for self-modification through open-ended exploration.

Darwin Godel Machine by Sakana AI

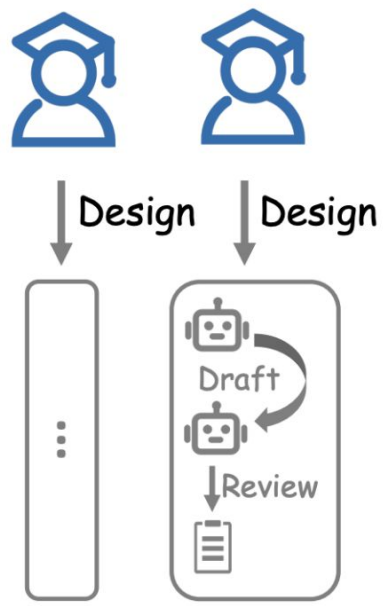


Darwin Godel Machine by Sakana AI

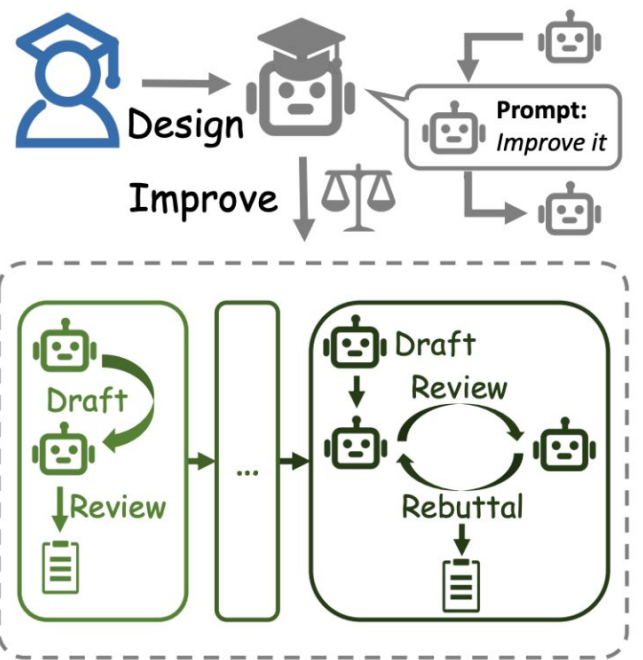


Gödel Agent

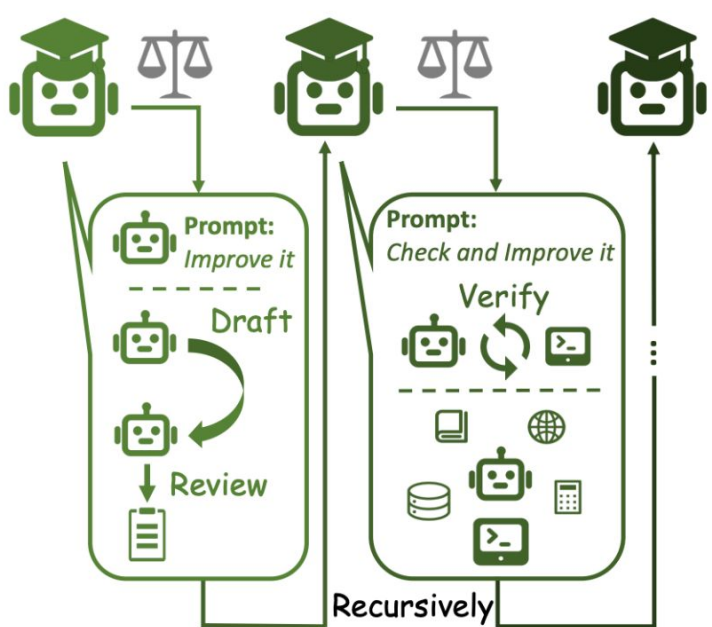
Hand-designed Agent



Meta-Learning Optimized Agent



Self-Referential Agent



Increasing degrees of freedom; Decreasing manual design; Fewer constraints and bottlenecks

Learnable Fixed Expert Meta Agent Agent Feedback Implementation

Huxley-Gödel Machine

HUXLEY-GÖDEL MACHINE: HUMAN-LEVEL CODING AGENT DEVELOPMENT BY AN APPROXIMATION OF THE OPTIMAL SELF-IMPROVING MACHINE

Wenyi Wang* **Piotr Piękos*** **Li Nanbo** **Firas Laakom** **Yimeng Chen**

Mateusz Ostaszewski **Mingchen Zhuge** **Jürgen Schmidhuber**

{wenyi.wang, piotr.piekos, nanbo.li, firas.laakom, yimeng.chen,
mateusz.ostaszewski, mingchen.zhuge, juergen.schmidhuber}@kaust.edu.sa

King Abdullah University of Science and Technology (KAUST)

Thuwal, Saudi Arabia

OpenEvolve

OpenEvolve: An Open Source Implementation of Google DeepMind's AlphaEvolve



Community Article

Published May 20, 2025



Asankhaya Sharma

[codelion](#)

Follow

```
python openevolve-run.py examples/function_minimization/initial_program.py \  
examples/function_minimization/evaluator.py \  
--config examples/function_minimization/config.yaml \  
--iterations 50
```

OpenEvolve: Rust sort example

https://github.com/algorithmicsuperintelligence/openevolve/tree/main/examples/rust_adaptive_sort

Rust Adaptive Sorting Evolution

This example demonstrates how to use OpenEvolve with the Rust programming language. The example focuses on evolving adaptive sorting algorithms that optimize their behavior based on input data characteristics, showcasing OpenEvolve's ability to work with compiled systems programming languages.

Files

- `initial_program.rs` : Starting Rust implementation with basic quicksort
- `evaluator.py` : Python evaluator that compiles and benchmarks Rust code
- `config.yaml` : Configuration optimized for performance-critical algorithm evolution
- `requirements.txt` : System dependencies and Python requirements

```

  examples
  > algotune
  > alphaevolve_math_problems
  > attention_optimization
  > circle_packing
  > circle_packing_with_artifacts
  > function_minimization
  > llm_prompt_optimization
  > lm_eval
  > mlx_metal_kernel_opt
  > online_judge_programming
  > r_robust_regression
  > rust_adaptive_sort
  > signal_processing
  > sldbench
  > symbolic_regression
  > web_scraper_optillm
  M↓ README.md
```

OpenEvolve: Rust sort example - config

```
# Configuration for Rust Adaptive Sorting Evolution

# General settings
max_iterations: 15
checkpoint_interval: 5
log_level: "INFO"
file_suffix: ".rs"

# LLM configuration
llm:
  models:
    - name: "gemini-3-flash-preview"
      weight: 0.7
      api_base: "https://generativelanguage.googleapis.com/v1beta/openai/"
      api_key: "${GEMINI_API_KEY}"

    - name: "claude-haiku-4-5-20251001"
      weight: 0.3
      api_base: "https://api.anthropic.com/v1"
      api_key: "${ANTHROPIC_API_KEY}"

  temperature: 0.7
  top_p: null
  max_tokens: 16384

# Custom system message for Rust performance programming
system_message: |
  You are an expert Rust systems programmer specializing in high-performance algorithms.
  Focus on creating adaptive sorting algorithms that can handle different data patterns.
```

```
# Prompt configuration
prompt:
  num_top_programs: 4

  # Include compilation errors and performance artifacts
  include_artifacts: true
  max_artifact_bytes: 8192

# Database configuration
database:
  population_size: 150
  num_islands: 4

  # Feature dimensions for sorting algorithms
  feature_dimensions:
    - "score" # Overall performance score
    - "performance_score" # Speed performance
    - "adaptability_score" # Adaptability to different data patterns
```

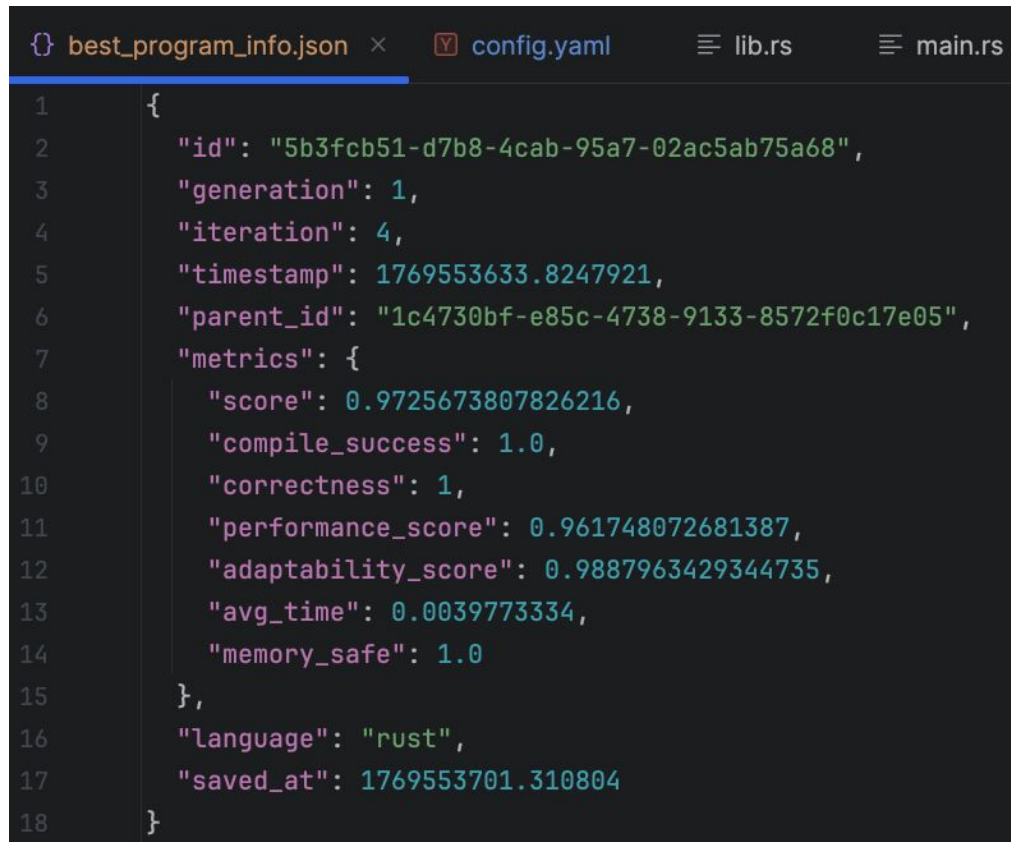
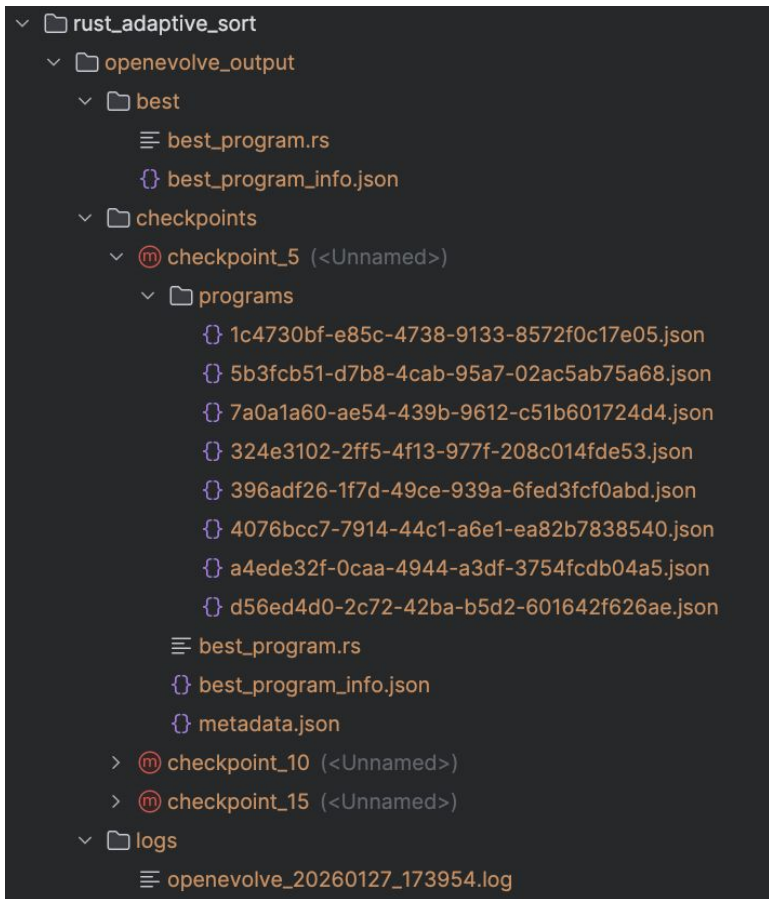
```
# Evaluator configuration
evaluator:
  timeout: 60 # Rust compilation can take time
  parallel_evaluations: 3
```

OpenEvolve: Rust sort example - evaluator.py

```
return EvaluationResult(  
    metrics={  
        "score": overall_score,  
        "compile_success": 1.0,  
        "correctness": correctness,  
        "performance_score": performance,  
        "adaptability_score": adaptability,  
        "avg_time": results["avg_time"],  
        "memory_safe": memory_safe,  
    },  
    artifacts={  
        "times": results["times"],  
        "all_correct": results["all_correct"],  
        "build_output": build_result.stdout,  
    },  
)
```

```
except subprocess.TimeoutExpired:  
    return EvaluationResult(  
        metrics={  
            "score": 0.0,  
            "compile_success": 0.0,  
            "correctness": 0.0,  
            "performance_score": 0.0,  
            "adaptability_score": 0.0,  
        },  
        artifacts={"error": "Timeout during evaluation"},  
    )
```

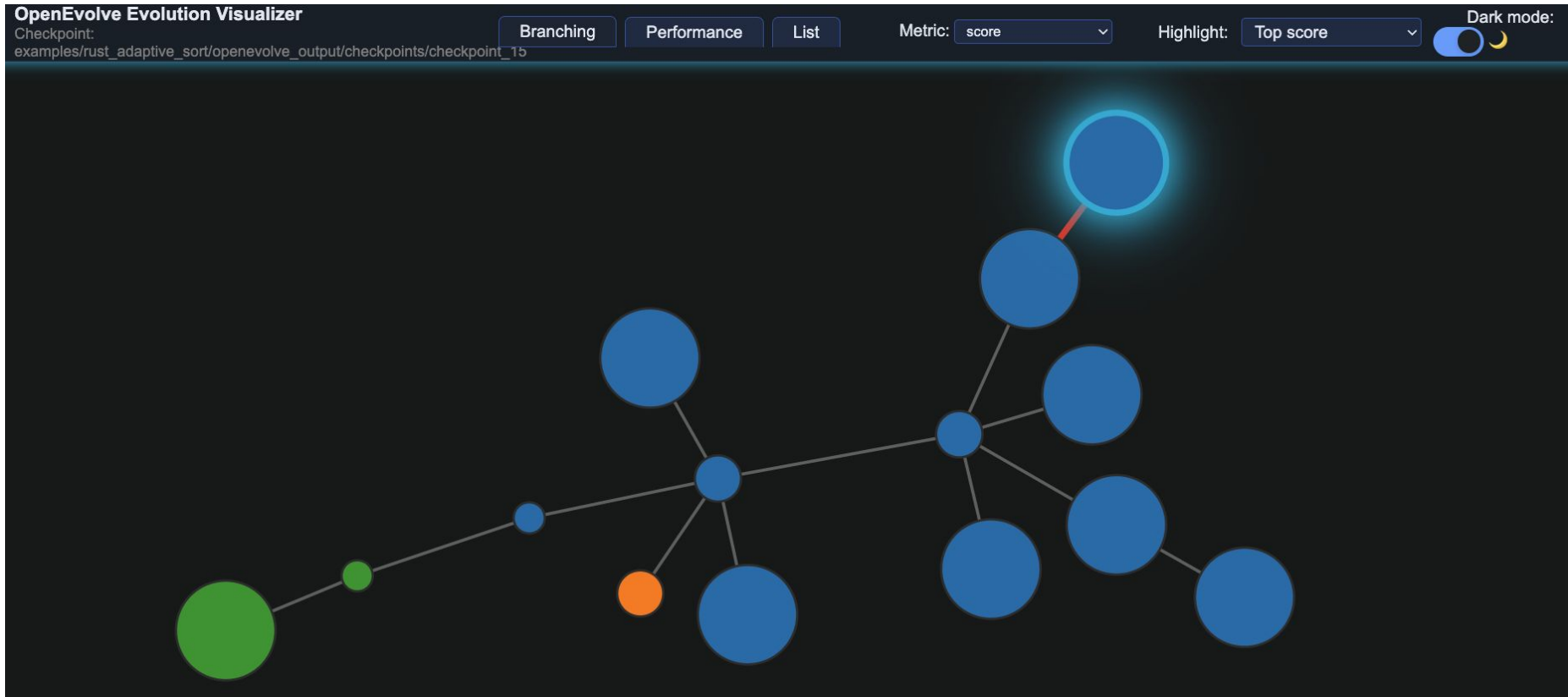
OpenEvolve: Rust sort example - generated artifacts



OpenEvolve: Rust sort example - best program

<pre>fn quicksort<T: Ord + Clone>(arr: &mut [T], low: usize, high: if low < high { let pivot_index = partition(arr, low, high); // Recursively sort elements before and after partitio if pivot_index > 0 { quicksort(arr, low, pivot_index - 1); } quicksort(arr, pivot_index + 1, high); } }</pre>	23 24 25 26 27 28 29 30 31 32 33 34	<pre>fn quicksort<T: Ord + Clone>(arr: &mut [T], low: usize, high: // Use insertion sort for small arrays (Hybrid Quicksort) const CUTOFF: usize = 16; if low < high { let len = high - low + 1; if len <= CUTOFF { insertion_sort(&mut arr[low..=high]); return; } let pivot_index = partition(arr, low, high);</pre>	23 24 25 26 27 28 29 30 31 32 33 34 35 36
---	--	---	--

OpenEvolve: Rust sort example - visualizer



Summary

- This isn't "scary self-improvement", but a tool for guided optimization
 - At least makes you think about your problem and evals more systematically :)
- With a human in the loop, it becomes automation for exploring ideas

Q&A

Reach out in

[https://www.linkedin.com/in/andimid/
dmytro.nikolaiev.ai@gmail.com](https://www.linkedin.com/in/andimid/dmytro.nikolaiev.ai@gmail.com)