# CodeEvolve: an Open Source Evolutionary Coding Agent for Algorithmic Discovery and Optimization

Inter Science Talks 2026

**Henrique Assumpção**[1,2], Diego Ferreira[1,2], Leandro Campos[1,2], Fabricio Murai[3]

[1]Inter Science - Inter&Co., Belo Horizonte, MG, Brazil
[2]Federal University of Minas Gerais, Belo Horizonte, MG, Brazil
[3]Worcester Polytechnic Institute, Worcester, MA, USA

April 2026

inter

DCC
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
UFMG

# Summary

# About

**A bit about me:**

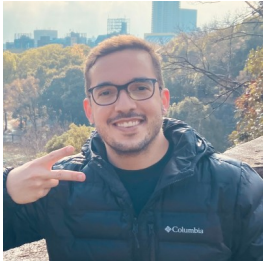
- ML Research Scientist at Inter&Co
- MS in Computer Science at UFMG
- Creator of CodeEvolve, contributor to OpenEvolve

**Co-Authors:**

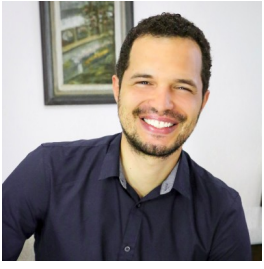
- Collaborative effort across Inter Science, UFMG and WPI.
- Specializing in NLP, Representation Learning and Recommender Systems.

H.A.

Fabricio Murai




Leandro Campos



Diego Ferreira

# Motivating Example

- Suppose that given matrices $A : n \times m, B : m \times k, C : n \times k$ stored as FP32, we want to compute

$$C' = \alpha \cdot AB + \beta \cdot C$$

as efficiently as possible (**S**ingle-precision **GE**neral **M**atrix **M**ultiply).

# Motivating Example

- Suppose that given matrices $A : n \times m, B : m \times k, C : n \times k$ stored as FP32, we want to compute

$$C' = \alpha \cdot AB + \beta \cdot C$$

  as efficiently as possible (**S**ingle-precision **GE**neral **M**atrix **M**ultiply).
- Optimizing this is an **extremely hard problem**.

# Motivating Example

- Suppose that given matrices $A : n \times m, B : m \times k, C : n \times k$ stored as FP32, we want to compute
$$C' = \alpha \cdot AB + \beta \cdot C$$
as efficiently as possible (**S**ingle-precision **GE**neral **M**atrix **M**ultiply).
- Optimizing this is an **extremely hard problem**.
- What happens if we ask an LLM to try to improve an algorithm for SGEMM?

# Motivating Example

## Naive SGEMM

```
if (global_row < M && global_col < N) {
    float dot = 0.0f;
    for (int k = 0; k < K; ++k) {
        dot += A[global_row * K + k] * B[k * N +
              global_col];
    }
    C[global_row * N + global_col] =
        alpha * dot + beta * C[global_row * N +
             global_col];
}
```

**LLM Prompt**
"Improve the SGEMM kernel."

## Block-Tiled SGEMM

```
for (int block_tile_start = 0; block_tile_start
     < K;
     block_tile_start += BSZ_K) {

    for (int i = thread_id; i < (BSZ_M * BSZ_K);
          i += nthreads_block) {
        As[r][c] = (g_r < M && g_c < K) ? A[g_r
             * K + g_c] : 0.0f;
    }

    for (int i = thread_id; i < (BSZ_K * BSZ_N);
          i += nthreads_block) {
        Bs[r][c] = (g_r < K && g_c < N) ? B[g_r
             * N + g_c] : 0.0f;
    }

    __syncthreads();

    for (int k = 0; k < BSZ_K; ++k) {
        dot += As[thread_row][k] * Bs[k][
             thread_col];
    }

    __syncthreads();
}
...
```

# Improving solutions

- In general, will this LLM-generated solution be close to the best-known solution?

# Improving solutions

- In general, will this LLM-generated solution be close to the best-known solution? **No.**

# Improving solutions

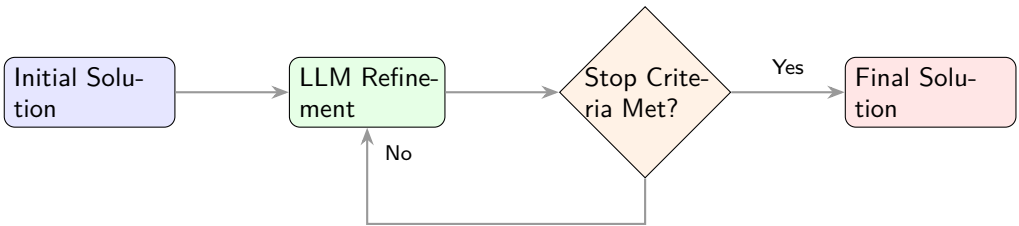
- In general, will this LLM-generated solution be close to the best-known solution? **No.**
- What can we do to improve the quality of the solution?

# Improving solutions

- In general, will this LLM-generated solution be close to the best-known solution? **No.**
- What can we do to improve the quality of the solution?
- Naive approach: Improving the prompt (Prompt engineering), Improving the original solution, Using a better LLM, etc.

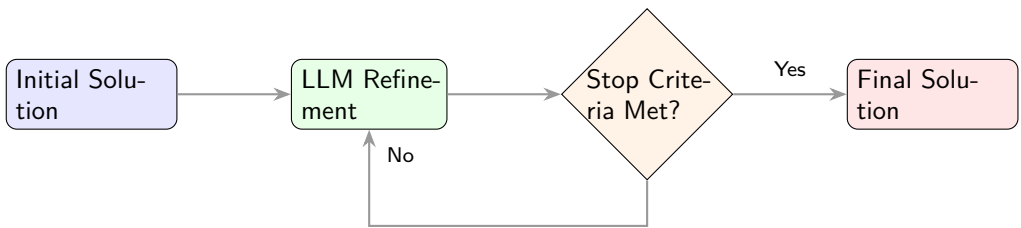
# Improving solutions

- In general, will this LLM-generated solution be close to the best-known solution? **No.**
- What can we do to improve the quality of the solution?
- Naive approach: Improving the prompt (Prompt engineering), Improving the original solution, Using a better LLM, etc.
- A less naive approach: simple loop of inputting the previous solution to the LLM until it becomes good enough, e.g.,



# Improving solutions

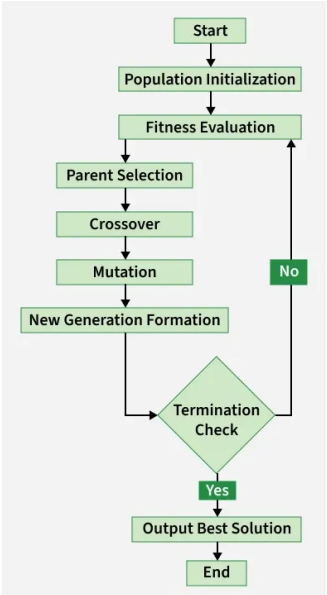
- In general, will this LLM-generated solution be close to the best-known solution? **No.**
- What can we do to improve the quality of the solution?
- Naive approach: Improving the prompt (Prompt engineering), Improving the original solution, Using a better LLM, etc.
- A less naive approach: simple loop of inputting the previous solution to the LLM until it becomes good enough, e.g.,




- Even better approach: use evolutionary algorithms!

# Evolutionary Algorithms

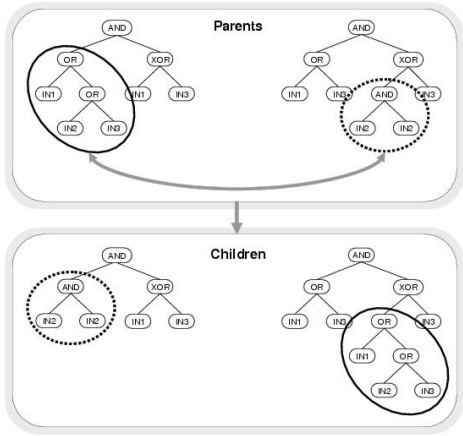
- **Origins:** Modern GP was popularized by John Koza in 1992 as an extension of Genetic Algorithms.
- **The Process:**
  - *Selection*
  - *Crossover*
  - *Mutation*
- **Goal:** Automated discovery of executable code that optimizes a given fitness function.



# Representing Code

**From Trees to Tokens**

- **Classic GP:** Programs are typically represented as **Abstract Syntax Trees (ASTs)**.
- **The Bottleneck:**
  - AST-based crossover/mutation is effective for small functions but scales poorly.
  - Hard to maintain long-range dependencies and global structure in large codebases.
- **The LLM Paradigm Shift:**
  - LLMs allow for **natural language-guided** mutations.
  - Agents can reason about code semantically rather than just manipulating syntax nodes.




Traditional Subtree Crossover Operation

# Evolutionary Loop for Agents

Solutions/Prompts $\rightarrow$ Selection $\rightarrow$ Crossover/Mutation (LLM) $\rightarrow$ Evaluation (Sandbox)

**Iterative Code Refinement**

- **1. Initialization:** A diverse population of initial solution attempts (prompts/code) is generated.
- **2. Selection:** Parent solutions are chosen based on some selection policy (e.g. fitness-based, uniformly random, etc).
- **3. Variation (LLM):** LLMs act as intelligent crossover and mutation operations, generating new solutions.
- **4. Fitness Evaluation:** New solutions are executed in a sandboxed environment, performance metrics and fitness are computed.
- **5. Termination:** The cycle repeats until the desired fitness is achieved or the computational budget is exhausted.

# LLMs for Code Generation

- **Initial Breakthroughs:** Early integration of LLMs in program synthesis (e.g., AlphaCode) proved models could generate competitive code.
- **The Evolutionary Shift:** Moving from simple zero-shot generation to "Evolution through Large Models" (ELM).
- **FunSearch & EoH:** Demonstrated that LLMs could discover new mathematical knowledge and optimize heuristics.
- **AlphaEvolve:** A critical milestone bridging the gap by synthesizing **end-to-end programs** at scale.

Google DeepMind

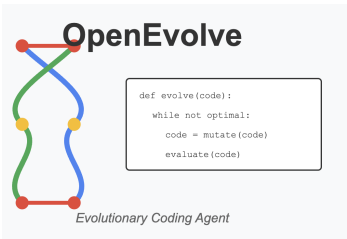
***AlphaEvolve*: A coding agent for scientific and algorithmic discovery**

Alexander Novikov*, Ngân Vũ*, Marvin Eisenberger*, Emilien Dupont*, Po-Sen Huang*, Adam Zsolt Wagner*, Sergey Shirobokov*, Borislav Kozlovskii*, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli and Matej Balog*
Google DeepMind[1]

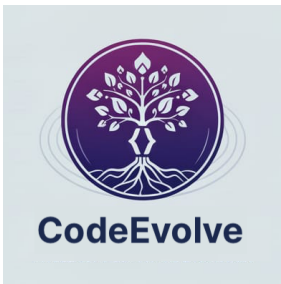
# OSS Frameworks

**Democratizing Algorithmic Discovery**

- Inspired by the success of AlphaEvolve, a wave of Open-Source Software (OSS) frameworks has emerged to provide transparency and reproducibility.
- **Some of the Frameworks:** OpenEvolve, ShinkaEvolve, ThetaEvolve, CodeEvolve, SkyDiscover, and many others.



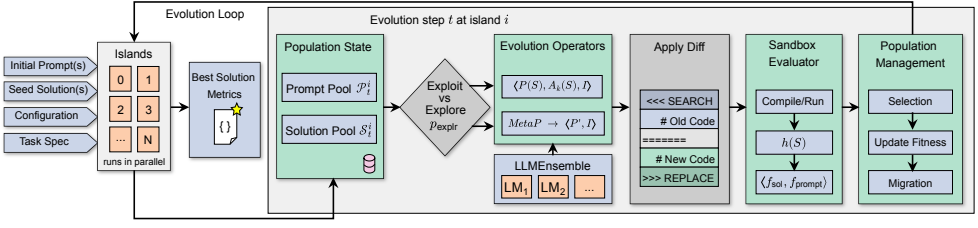




# Overview of CodeEvolve

## A Transparent Framework for Meta-Optimization

- **Island-Based GA:** Maintains diversity through isolated populations and parallel search trajectories.
- **Weighted LLM Ensemble:** Dynamically selects the best-performing models for specific mutation tasks.
- **Modular Operators:**
  - *Inspiration-based Crossover:* Recombines logic from top-tier "parent" programs.
  - *Meta-Prompting:* Evolves the instructions themselves to guide the search.
  - *Depth-based Refinement:* Targets specific code segments for local exploitation.




Architecture of the CodeEvolve Evolutionary Pipeline

# LLM Backbones

**Comparing Proprietary and Open-Weight Models**

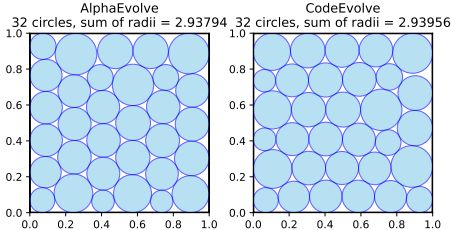
- Our experiments evaluate the scalability and reasoning capabilities of:
- **Gemini 2.5 (Flash/Pro):** Serving as the high-throughput, closed-source baseline for complex reasoning.
- **Qwen3-Coder-30B:** Representing the state-of-the-art in open-weight coding models.
- **Key finding:** Open-weight models often match or exceed closed-source performance at significantly lower compute costs.


Google Gemini 2.5


Alibaba Qwen3-Coder30B

# Benchmark Problems

- **Geometric Packing:** Packing circles and hexagons in order to optimize a given metric.
- **Min-Max Distance Ratio:** Placing points in $d$-dimensional space to minimize the ratio of maximum to minimum distance.
- **Autocorrelation Inequalities:** Constructing optimal step-functions for convolution bounds in additive combinatorics.
- **Evaluation:** Programs are executed in a sandbox and scored against rigid mathematical fitness signals.



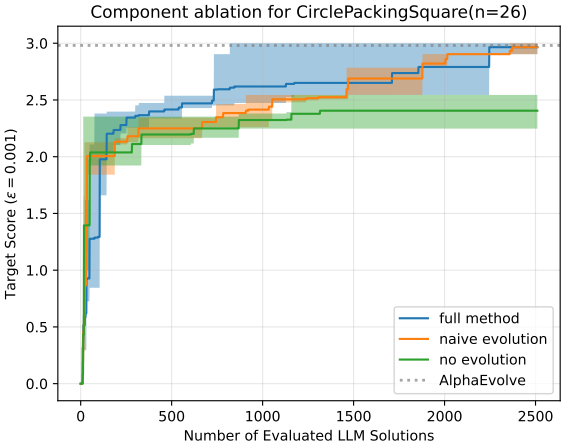
# Overview of Results

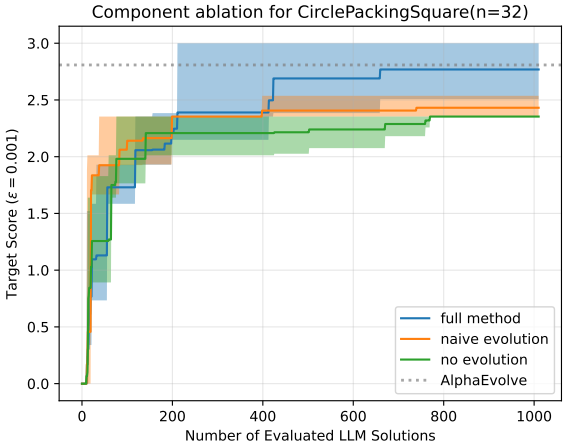
| Problem | AlphaEvolve | CodeEvolve | |
|---|---|---|---|
| | | Qwen3-30B | Gemini 2.5 |
| `CirclePackingSq` ($n = 26$) ($\uparrow$) | 2.63586 | **2.63598** | 2.63597 |
| `CirclePackingSq` ($n = 32$) ($\uparrow$) | 2.93794 | **2.93956** | 2.93950 |
| `CirclePackingRect` ($n = 21$) ($\uparrow$) | **2.36583** | 2.36339 | **2.36583** |
| `HexagonPacking` ($n = 11$) ($\downarrow$) | **3.93009** | 4.02786 | 3.93794 |
| `HexagonPacking` ($n = 12$) ($\downarrow$) | **3.94191** | 4.01092 | 4.00001 |
| `MinMaxDist` ($n = 16, d = 2$) ($\downarrow$) | 12.88927 | 12.88925 | **12.88923** |
| `MinMaxDist` ($n = 14, d = 3$) ($\downarrow$) | 4.16585 | 4.16765 | **4.16579** |
| `FirstAutocorrIneq` ($\downarrow$) | **1.50316** | 1.51343 | 1.55438 |
| `SecondAutocorrIneq` ($\uparrow$) | **0.96102** | 0.88110 | 0.87067 |

# Overview of Results

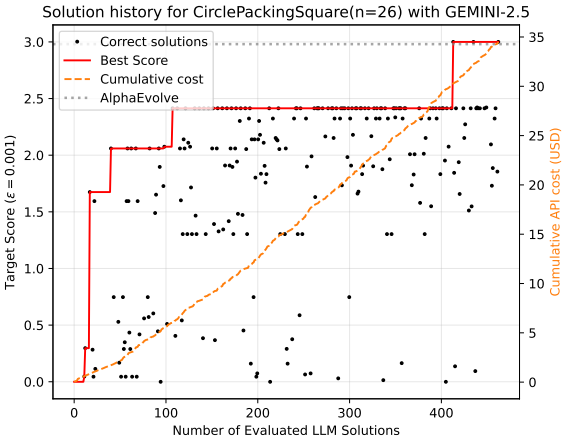
| Problem | ShinkaEvolve | | OpenEvolve | | CodeEvolve | |
|---|---|---|---|---|---|---|
| | Mean | Best | Mean | Best | Mean | Best |
| `CirclePackingSquare`$(n = 26)(\uparrow)$ | 2.47568 | 2.61551 | 2.54962 | 2.6245 | **2.63576** | **2.63598** |
| `CirclePackingSquare`$(n = 32)(\uparrow)$ | 2.53205 | 2.69473 | 2.84897 | 2.93156 | **2.93751** | **2.93957** |
| `CirclePackingRect`$(n = 21)(\uparrow)$ | 2.21694 | 2.32701 | 2.28131 | **2.36513** | **2.36212** | 2.36465 |
| `HexagonPacking`$(n = 11)(\downarrow)$ | 5.55234 | 5.00000 | 4.30696 | 4.20679 | **4.02491** | **4.01377** |
| `HexagonPacking`$(n = 12)(\downarrow)$ | 5.07507 | 4.38608 | **4.01991** | **4.0001** | 4.10841 | 4.05198 |
| `MinimizeMaxMinDist`$(n = 16, d = 2)(\downarrow)$ | 13.48234 | 13.11627 | 12.97046 | 12.94042 | **12.91569** | **12.90274** |
| `MinimizeMaxMinDist`$(n = 14, d = 3)(\downarrow)$ | 4.37160 | 4.21087 | **4.18983** | 4.1714 | 4.26257 | **4.16765** |
| `FirstAutocorrIneq`$(\downarrow)$ | **1.52173** | **1.51556** | 1.58031 | 1.5751 | 1.54696 | 1.51642 |
| `SecondAutocorrIneq`$(\uparrow)$ | 0.82010 | 0.84646 | 0.84634 | 0.86086 | **0.86533** | **0.90114** |

# Does Evolution Matter?

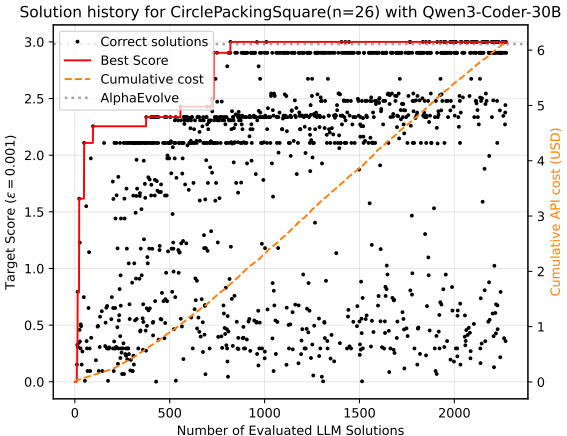


# Sample Efficiency




Gemini 2.5 Convergence Profile




Qwen3-Coder-30B Convergence Profile

# Sample Efficiency

| Problem | Qwen3-Coder-30B | | GEMINI-2.5 | |
|---|---|---|---|---|
| | **Cost (USD)** | **Time (Hours)** | **Cost (USD)** | **Time (Hours)** |
| `CirclePackingSquare`$(n = 26)$ | 6.2 | 6.4 | 34.5 | 6.1 |
| `CirclePackingSquare`$(n = 32)$ | 2.1 | 2.2 | 17.2 | 7.5 |
| `CirclePackingRect`$(n = 21)$ | 10.2 | 12 | 67.5 | 11.1 |
| `HexagonPacking`$(n = 11)$ | 9.1 | 9.8 | 73.7 | 17.5 |
| `HexagonPacking`$(n = 12)$ | 7.4 | 8.5 | 77.4 | 15.6 |
| `MinimizeMaxMinDist`$(n = 16, d = 2)$ | 15.33 | 25.7 | 51.5 | 15.7 |
| `MinimizeMaxMinDist`$(n = 14, d = 3)$ | 9.6 | 10.9 | 54.4 | 13.1 |
| `FirstAutocorrIneq` | 23.1 | 54 | 70.8 | 17.3 |
| `SecondAutocorrIneq` | 27.2 | 28.4 | 66.7 | 18.7 |

# Cited by Leading Industry Research Teams




# Cited by Top Academic Institutions








# Conclusion

**The Power of Evolution-Augmented LLMs**

- **Synergy is Key:** Combining evolutionary search with LLMs creates a discovery engine far more powerful than zero-shot prompting alone.
- **Evolution as an Equalizer:** Experiments show that the evolutionary loop allows OSS models like Qwen to match or surpass proprietary ones.
- **Strategic Independence:** This paradigm reduces dependency on closed-source APIs, offering a path toward high-performance, private, and reproducible automated discovery.
- **The Loop $>$ The Model:** The architecture of the search process is often more critical for discovery than the raw size of the underlying LLM.

# Next Steps

**Shape the Future of Algorithmic Discovery**

- **Open-Source Contributions Wanted:**
  - Help us scale the core pipeline on GitHub (operators, backbones).
  - Implement new sandboxed evaluation tools for diverse languages.
- **Invitation to Inter Employees:**
  - Bring your domain-specific optimization or heuristic bottlenecks to us.
  - Leverage the framework internally to automate your discovery tasks.
- **Expanding the Benchmark Ecosystem:** Integrating continuous control, financial risk modeling, and hyperparameter tuning environments.

# Thank you!

## Questions?

*henrique.soares@inter.co*
*henriqueassumpcao.github.io*

CodeEvolve GitHub

AI-Driven Discovery of Algorithms
2026 Conference