OpenEvolve: Towards Open Evolutionary Agents

Asankhaya Sharma

Co-Founder & CTO
Patched.Codes

APPLIED SUMMIT AI

Presented by John Snow LABS

Google DeepMind 2025-5-16

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]

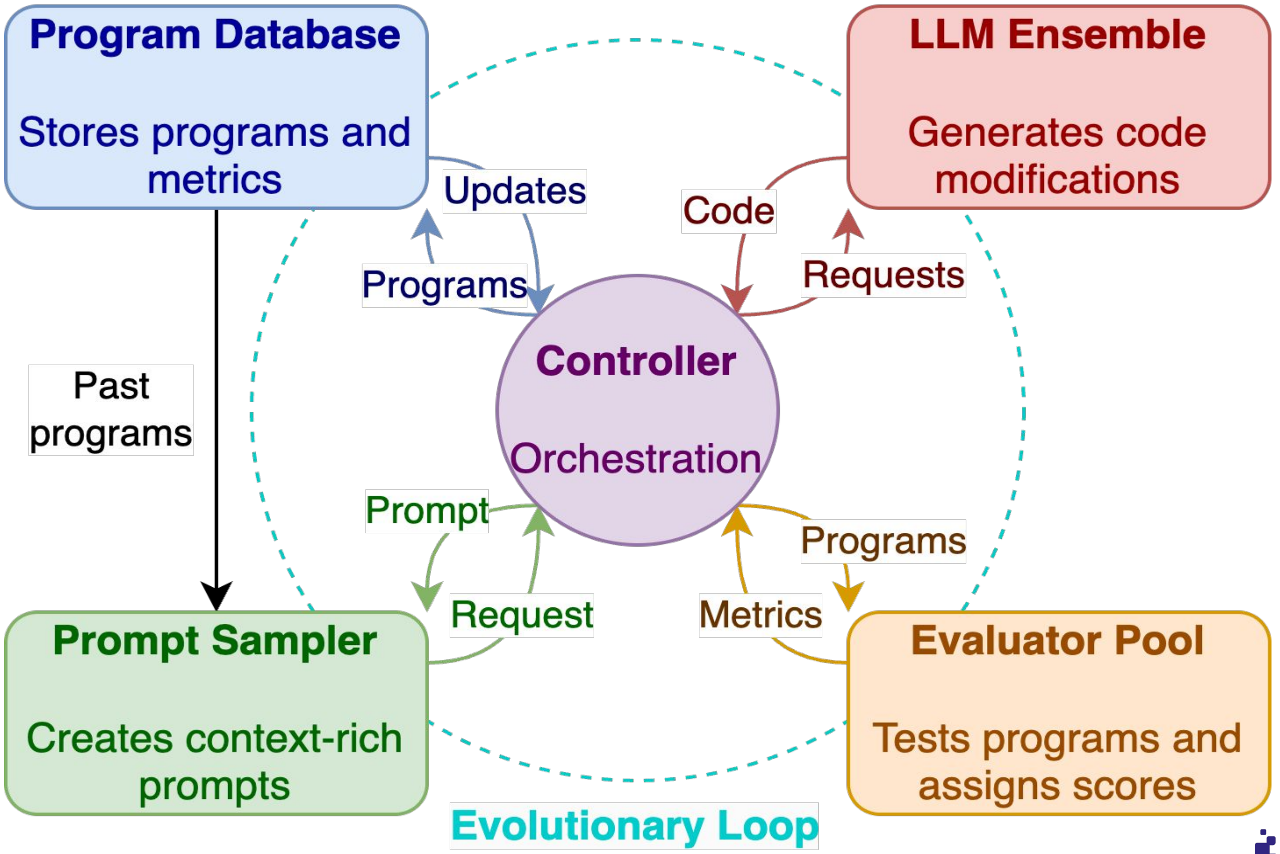
In this white paper, we present *AlphaEvolve*, an evolutionary coding agent that substantially enhances capabilities of state-of-the-art LLMs on highly challenging tasks such as tackling open scientific problems or optimizing critical pieces of computational infrastructure. *AlphaEvolve* orchestrates an autonomous pipeline of LLMs, whose task is to improve an algorithm by making direct changes to the code. Using an evolutionary approach, continuously receiving feedback from one or more evaluators, *AlphaEvolve* iteratively improves the algorithm, potentially leading to new scientific and practical discoveries. We demonstrate the broad applicability of this approach by applying it to a number of important computational problems. When applied to optimizing critical components of large-scale computational stacks at Google, *AlphaEvolve* developed a more efficient scheduling algorithm for data centers, found a functionally equivalent simplification in the circuit design of hardware accelerators, and accelerated the training of the LLM underpinning *AlphaEvolve* itself. Furthermore, *AlphaEvolve* discovered novel, provably correct algorithms that surpass state-of-the-art solutions on a spectrum of problems in mathematics and computer science, significantly expanding the scope of prior automated discovery methods (Romera-Paredes et al., 2023). Notably, *AlphaEvolve* developed a search algorithm that found a procedure to multiply two 4×4 complex-valued matrices using 48 scalar multiplications; offering the first improvement, after 56 years, over Strassen's algorithm in this setting. We believe *AlphaEvolve* and coding agents like it can have a significant impact in improving solutions of problems across many areas of science and computation.

OpenEvolve

Google DeepMind published a paper on 18th May 2025 about the AlphaEvolve system which described several results using an evolutionary coding agent that enhanced the capabilities of frontier models.

We decided to replicate the work and build a similar system and release it as OSS. This was the genesis of OpenEvolve.

OpenEvolve Architecture



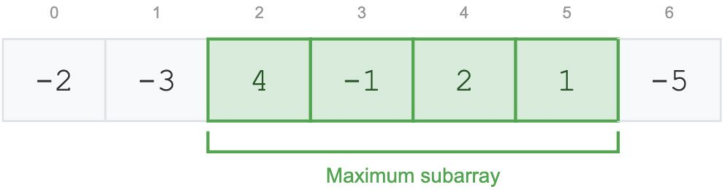
Asynchronous pipeline optimized for maximum throughput

APPLIED SUMMIT AI

Example Problem: Maximum Subarray Sum

Find the contiguous subarray with the largest sum

Input Array:



Maximum Sum = 6

Subarray [4, -1, 2, 1] has the largest sum

4 + (-1) + 2 + 1 = 6

How can we solve this efficiently?

Starting Point: Naive Solution

```python
def max_subarray_sum(arr):
 n = len(arr)
 max_sum = float('-inf')

 # Try every possible subarray
 for i in range(n):
    for j in range(i, n):
       current_sum = 0
       for k in range(i, j + 1):
          current_sum += arr[k]

       max_sum = max(max_sum, current_sum)

 return max_sum
```

Time: $O(n^3)$ - Very Slow!

Three nested loops checking every subarray

For 1000 elements: ~1 billion operations!

OpenEvolve: Just 3 Inputs

1 Initial Code

```
def max_subarray():
# O(n³) solution
...
```

2 Evaluator

```
✓ Correctness
✓ Speed
✓ Score: 0-100
```

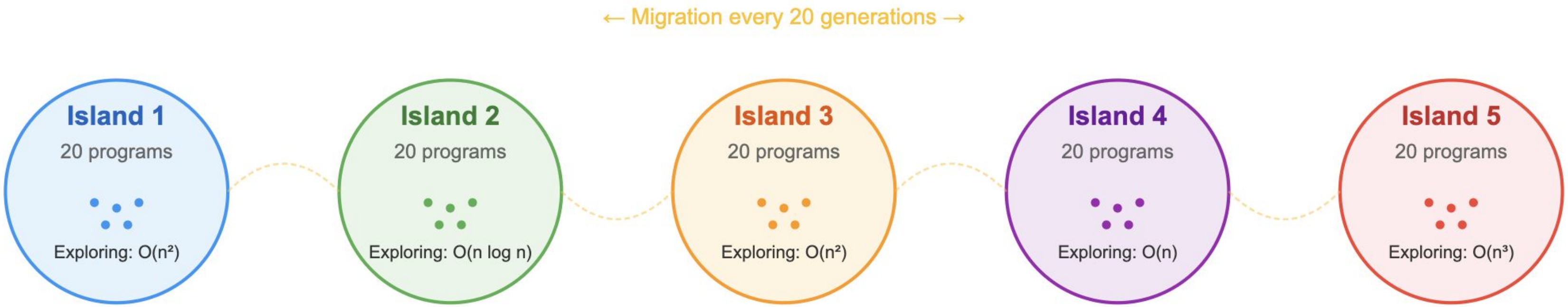
3 Config

```
LLM: Gemini-2.5
Islands: 5
Iterations: 1000
```

That's all! Evolution begins...

Parallel Evolution with Islands

5 populations evolve independently, preventing getting stuck

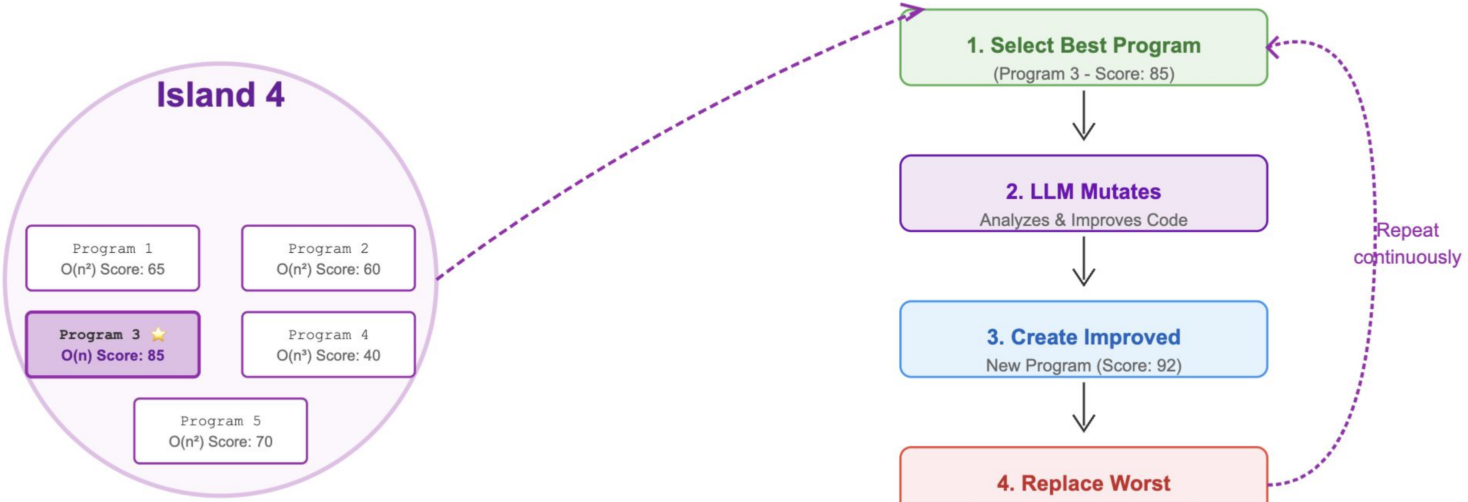


- ✓ Each island has its own evolutionary loop
- ✓ Best solutions spread through migration
- ✓ Diversity prevents premature convergence

APPLIED SUMMIT AI

Inside Island 4: Evolution in Action

Each island runs its own evolutionary loop



💡 Each island independently evolves using LLM-guided improvements

One Iteration: LLM Improvement

How Island 4's best program gets improved

System Prompt to LLM:

"Improve this O(n) solution. Fix any bugs. Optimize if possible."

Selected from Island 4

Score: 85/100

```
def max_subarray(arr):
max_sum = arr[0]
current = arr[0]
for num in arr[1:]:
current = current + num
```

⚠ *Bug: Never resets negative sums*

LLM

Gemini-2.5

Improved Version

Score: 100/100 ✓

```
def max_subarray(arr):
max_sum = arr[0]
current = arr[0]
for num in arr[1:]:
current = max(num, current+num)
```

✓ *Fixed: Kadane's algorithm!*

LLM Analysis:

"The current sum never resets when it becomes negative."

"Using max(num, current+num) implements Kadane's algorithm correctly."

"This handles negative numbers and maintains O(n) complexity."

This improved program replaces the worst program in Island 4

After 1000 Iterations: Optimal Solution!

🏆

Final Evolved Solution - Kadane's Algorithm

```
def max_subarray_sum(arr):
# Kadane's Algorithm - O(n) time, O(1) space
max_sum = arr[0]
current_sum = arr[0]
for num in arr[1:]:
    current_sum = max(num, current_sum + num)
    max_sum = max(max_sum, current_sum)
return max_sum
```

Started: $O(n^3)$
~1 billion ops for n=1000

→

Evolved: $O(n)$
1000 ops for n=1000

1,000,000x faster!

APPLIED SUMMIT AI

Circle Packing Problem (n=26)

The circle packing problem involves placing n non-overlapping circles inside a container (in this case, a unit square) to optimize a specific metric.

For this example:

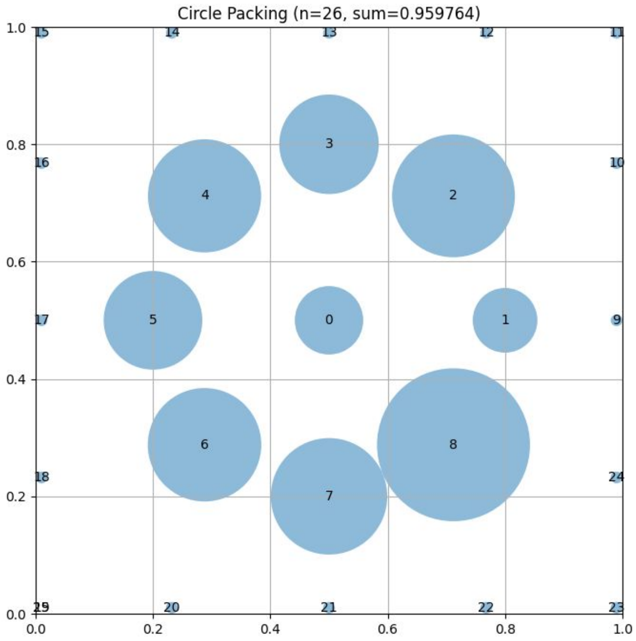
- We pack exactly 26 circles
- Each circle must lie entirely within the unit square
- No circles may overlap
- We aim to maximize the sum of all circle radii

According to the AlphaEvolve paper, a solution with a sum of radii of approximately 2.635 is achievable for n=26.

Our goal was to match or exceed this result.

Initial Program

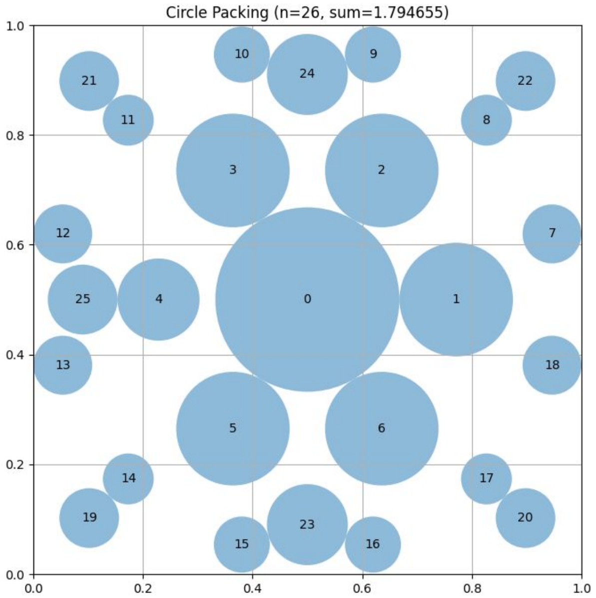
```
# Initial attempt
# Place a large circle in the center
centers[0] = [0.5, 0.5]

# Place 8 circles around it in a ring
for i in range(8):
    angle = 2 * np.pi * i / 8
    centers[i + 1] = [0.5 + 0.3 *
np.cos(angle), 0.5 + 0.3 * np.sin(angle)]

# Place 16 more circles in an outer ring
for i in range(16):
    angle = 2 * np.pi * i / 16
    centers[i + 9] = [0.5 + 0.7 *
np.cos(angle), 0.5 + 0.7 * np.sin(angle)]
```



APPLIED SUMMIT AI

Evolved Program

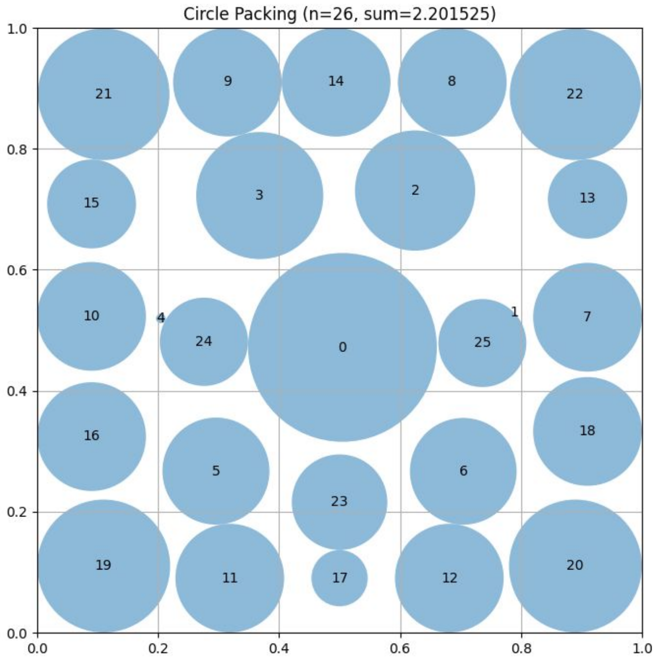
```
# Generation 10
# Parameters for the arrangement (fine-tuned)
r_center = 0.1675  # Central circle radius

# 1. Place central circle
centers[0] = [0.5, 0.5]
radii[0] = r_center

# 2. First ring: 6 circles in hexagonal
arrangement
r_ring1 = 0.1035
ring1_distance = r_center + r_ring1 + 0.0005  #
Small gap for stability
for i in range(6):
    angle = 2 * np.pi * i / 6
    centers[i+1] = [
        0.5 + ring1_distance * np.cos(angle),
        0.5 + ring1_distance * np.sin(angle)
    ]
    radii[i+1] = r_ring1
```



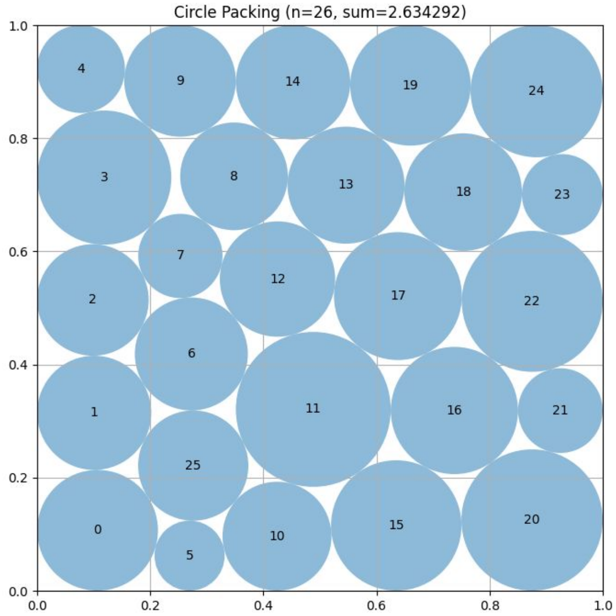
APPLIED SUMMIT AI

After 200 iterations

```
# Generation 100
# Row 1: 5 circles
centers[0] = [0.166, 0.166]
centers[1] = [0.333, 0.166]
centers[2] = [0.500, 0.166]
centers[3] = [0.667, 0.166]
centers[4] = [0.834, 0.166]

# Row 2: 6 circles (staggered)
centers[5] = [0.100, 0.333]
centers[6] = [0.266, 0.333]
# ... additional circles
```

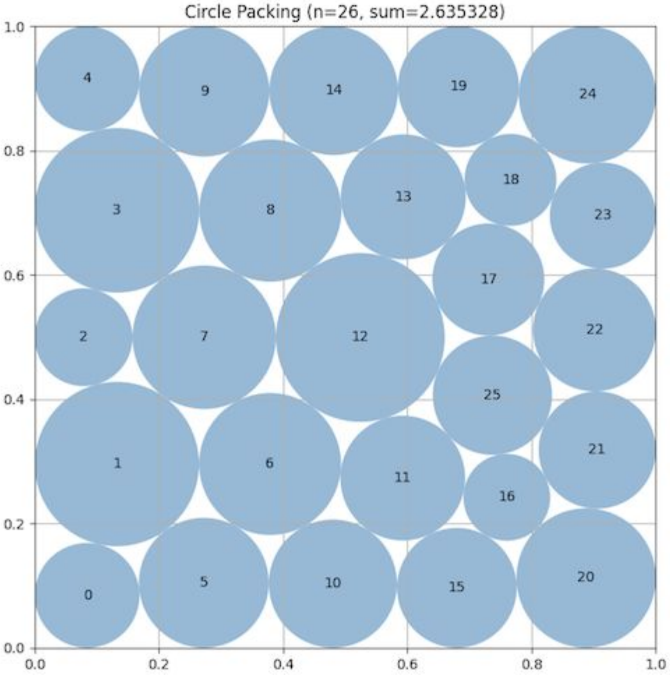


```python
# Final solution with scipy.optimize
def construct_packing():
    # ... initialization code ...

    # Objective function: Negative sum of radii (to
maximize)
    def objective(x):
        centers = x[:2*n].reshape(n, 2)
        radii = x[2*n:]
        return -np.sum(radii)

    # Constraint: No overlaps and circles stay within
unit square
    def constraint(x):
        centers = x[:2*n].reshape(n, 2)
        radii = x[2*n:]

        # Overlap constraint
        overlap_constraints = []
        for i in range(n):
            for j in range(i + 1, n):
                dist = np.sqrt(np.sum((centers[i] -
centers[j])**2))
                overlap_constraints.append(dist -
(radii[i] + radii[j]))
        # ... boundary constraints ...

    # Optimization using SLSQP
    result = minimize(objective, x0, method='SLSQP',
bounds=bounds, constraints=constraints)
```

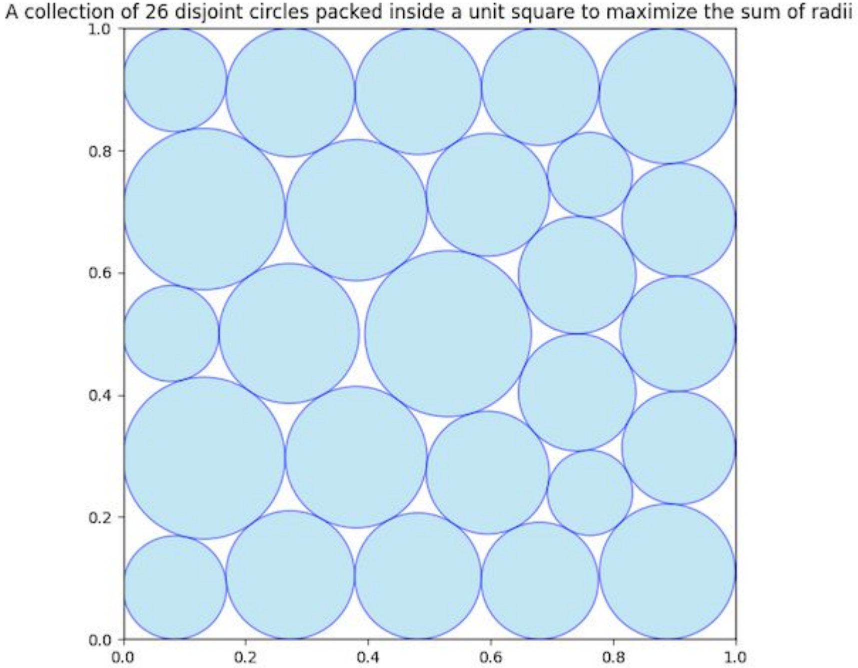


APPLIED SUMMIT AI

OpenEvolve



AlphaEvolve



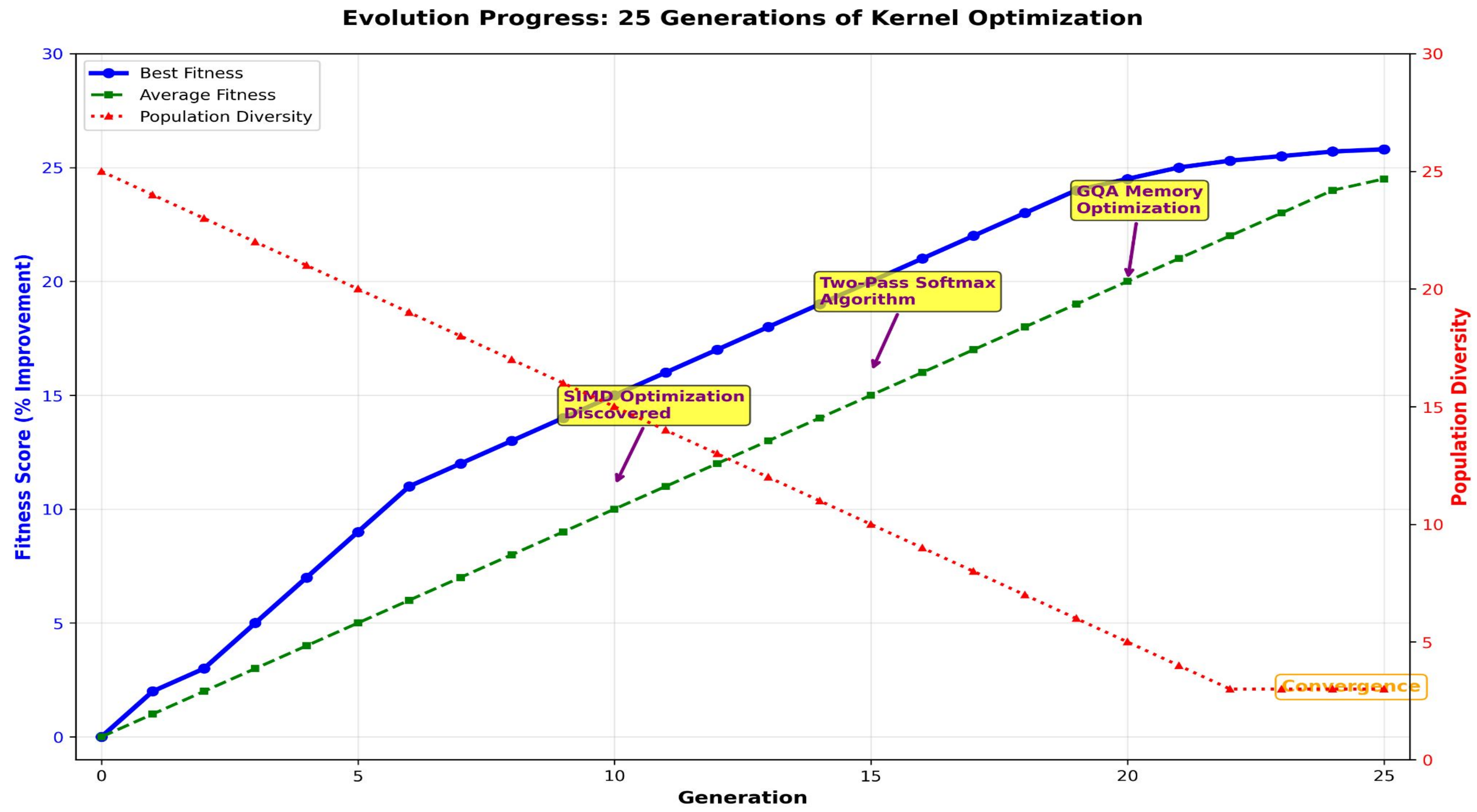
MLX Metal Kernel for Transformer Attention

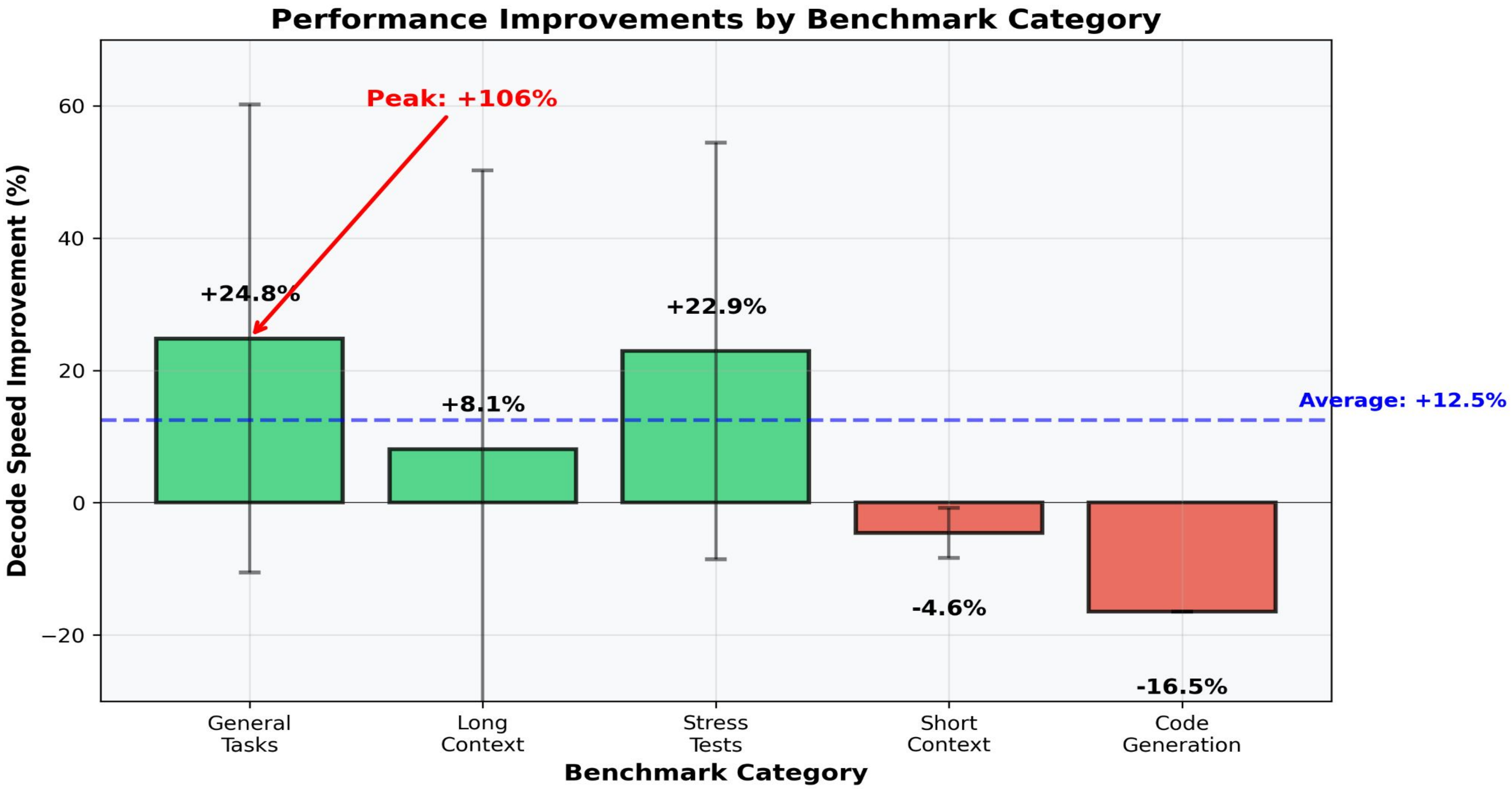
The Challenge

- **Target**: Qwen3-0.6B with Grouped Query Attention (40:8 head ratio)
- **Hardware**: Apple Silicon M-series GPUs with unified memory
- **Baseline**: MLX's highly optimized scaled_dot_product_attention
- **Goal**: Outperform expert-engineered kernel through automated discovery

Why This is Hard

- MLX is already highly optimized by Apple's engineers
- Attention kernels are performance-critical
- Apple Silicon has unique architectural features


Evolution Progress: 25 Generations of Kernel Optimization
Best Fitness
Average Fitness
Population Diversity
SIMD Optimization Discovered
Two-Pass Softmax Algorithm
GQA Memory Optimization
Convergence
Fitness Score (% Improvement)
Population Diversity
Generation
0
5
10
15
20
25
30



APPLIED SUMMIT AI

AlgoTune

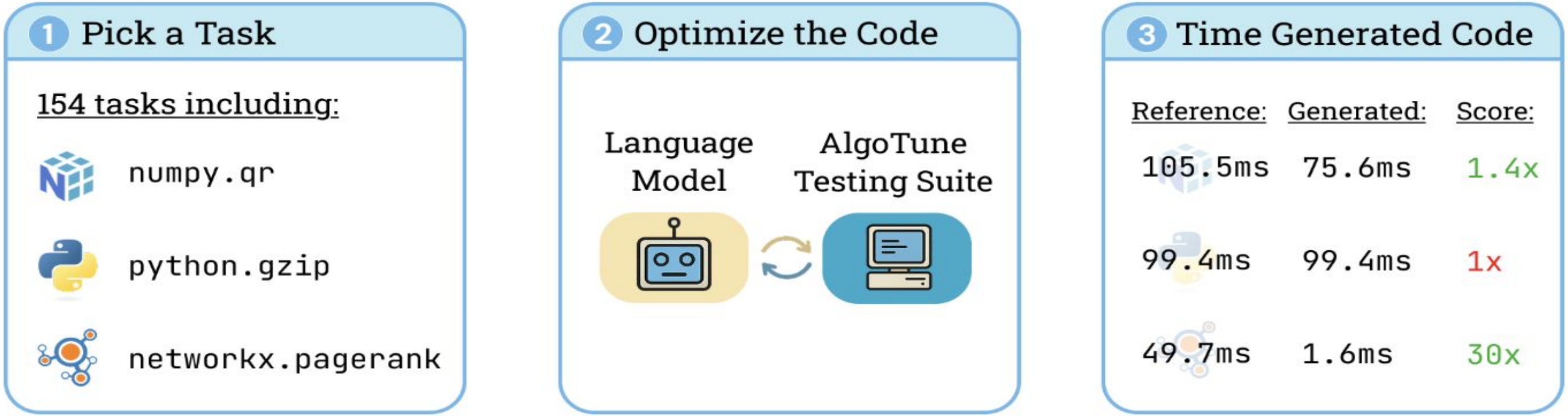
Can Language Models Speed Up General-Purpose Numerical Programs?

UNIVERSITÄT TÜBINGEN

Ori Press Brandon Amos Haoyu Zhao Yikai Wu Samuel K. Ainsworth Dominik Krupke Patrick Kidger
Touqir Sajed Bartolomeo Stellato Jisun Park Nathanael Bosch Eli Meril Albert Steppi
Arman Zharmagambetov Fangzhao Zhang David Pérez-Piñeiro Alberto Mercurio Ni Zhan
Talor Abramovich Kilian Lieret Hanlin Zhang Shirley Huang Matthias Bethge Ofir Press

PRINCETON UNIVERSITY

Paper Code



OpenEvolve

AlgoTuner

o4-mini	1.72x
DeepSeek R1	1.70x
GPT-5	1.67x
GLM-4.5	1.52x
Gemini 2.5 Pro	1.51x

APPLIED SUMMIT AI

Things to watch out for!

- Choosing the right abstraction at which to do the evolutionary search
- Preventing and allowing the use of existing libraries and APIs
- Guiding a population of candidate programs via prompting v/s a single program
- Robust cascading evaluations
- Requires human ingenuity in formulating the problem

New paradigm

- For inference time scaling of LLMs
- Distinct from existing sequential or parallel test time computing approaches
- Combines genetic algorithms driven search with LLMs for evolutionary coding agents
- Distill evolutionary agents to next version of base LLMs

Thank You!

- Questions?
- Links
  - OpenEvolve - https://github.com/codelion/openevolve
  - EvoVisual - https://evovisual-advanced-evolutionary-concepts-577160257370.us-west1.run.app/
  - OpenEvolve: An Open Source Implementation of Google DeepMind's AlphaEvolve - https://huggingface.co/blog/codelion/openevolve
  - Automated Discovery of High-Performance GPU Kernels with OpenEvolve - https://huggingface.co/blog/codelion/openevolve-gpu-kernel-discovery
  - Towards Open Evolutionary Agents - https://huggingface.co/blog/driaforall/towards-open-evolutionary-agents

APPLIED SUMMIT AI