OpenEvolve: Towards Open Evolutionary Agents

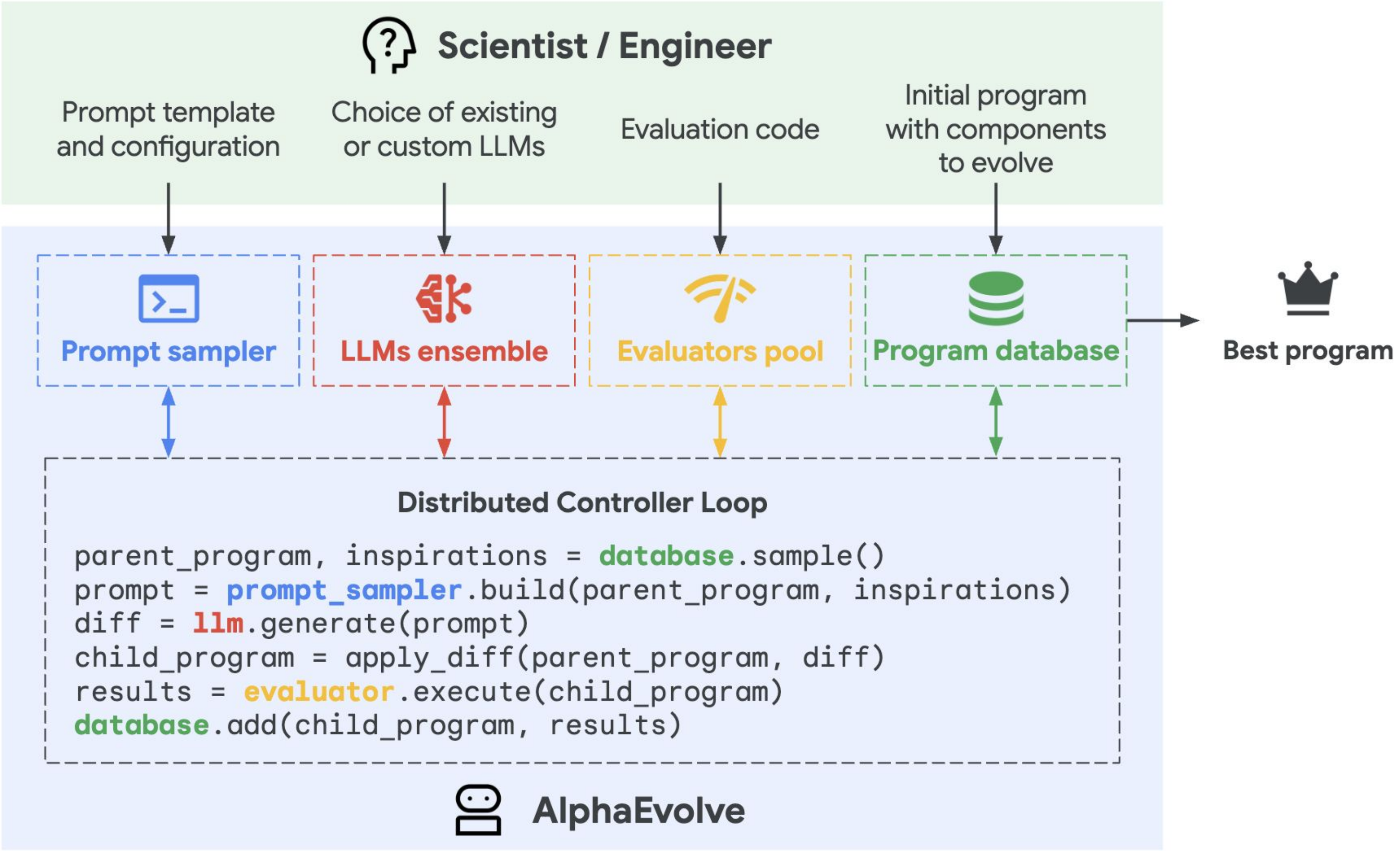
Asankhaya Sharma

https://github.com/codelion/openevolve

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.

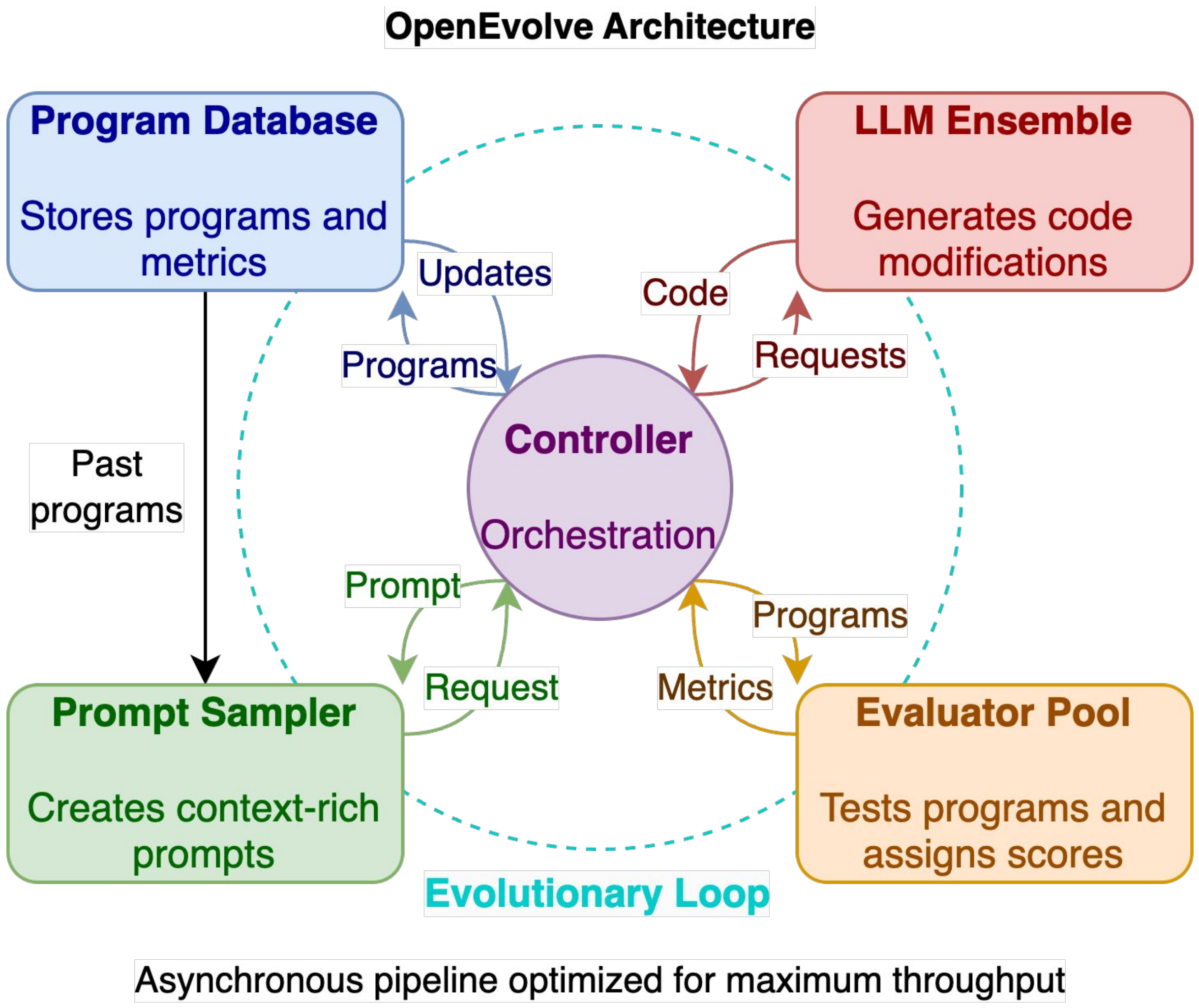


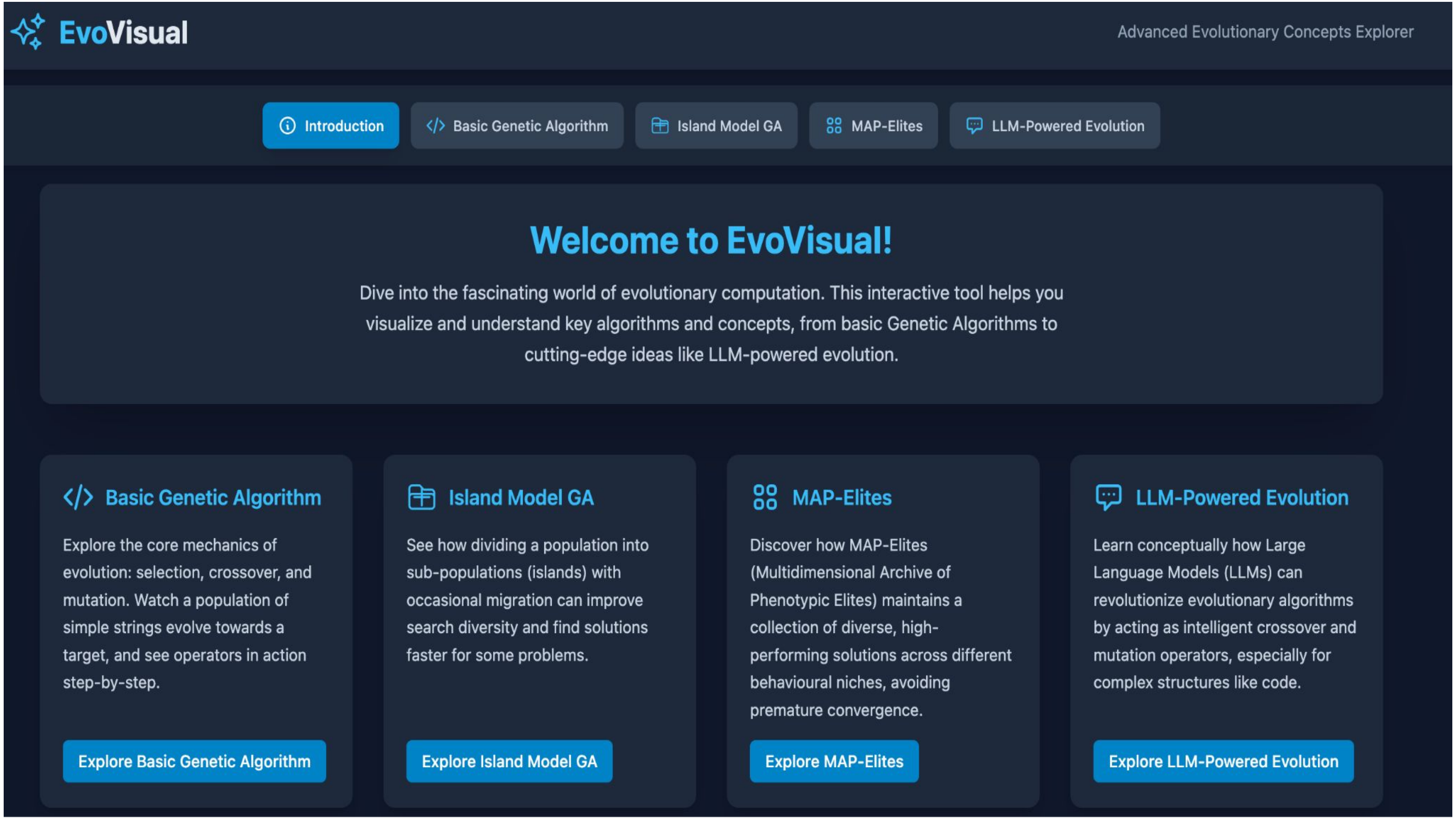
Scientist / Engineer
Prompt template and configuration
Choice of existing or custom LLMs
Evaluation code
Initial program with components to evolve
Prompt sampler
LLMs ensemble
Evaluators pool
Program database
Best program
Distributed Controller Loop
parent_program, inspirations = database.sample()
prompt = prompt_sampler.build(parent_program, inspirations)
diff = llm.generate(prompt)
child_program = apply_diff(parent_program, diff)
results = evaluator.execute(child_program)
database.add(child_program, results)
AlphaEvolve

OpenEvolve

```
def evolve(code):
  while not optimal:
    code = mutate(code)
    evaluate(code)
```

Evolutionary Coding Agent


OpenEvolve Architecture
Program Database
Stores programs and metrics
LLM Ensemble
Generates code modifications
Updates
Code
Programs
Requests
Controller
Orchestration
Past programs
Prompt
Programs
Request
Metrics
Prompt Sampler
Creates context-rich prompts
Evaluator Pool
Tests programs and assigns scores
Evolutionary Loop
Asynchronous pipeline optimized for maximum throughput



https://evovisual-advanced-evolutionary-concepts-577160257370.us-west1.run.app

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:

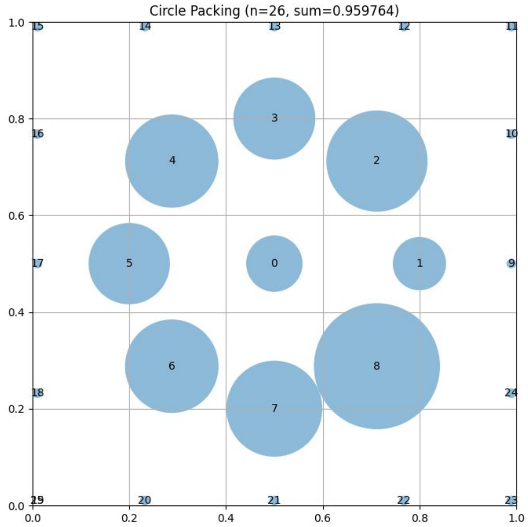
- We pack exactly 26 circles
- Each circle must lie entirely within the unit square
- No circles may overlap
- We aim to maximize the sum of all circle radii

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

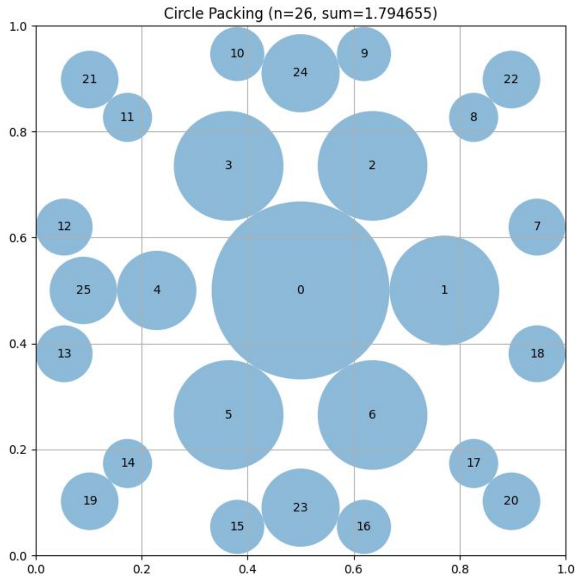
Our goal was to match or exceed this result.

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)]
```

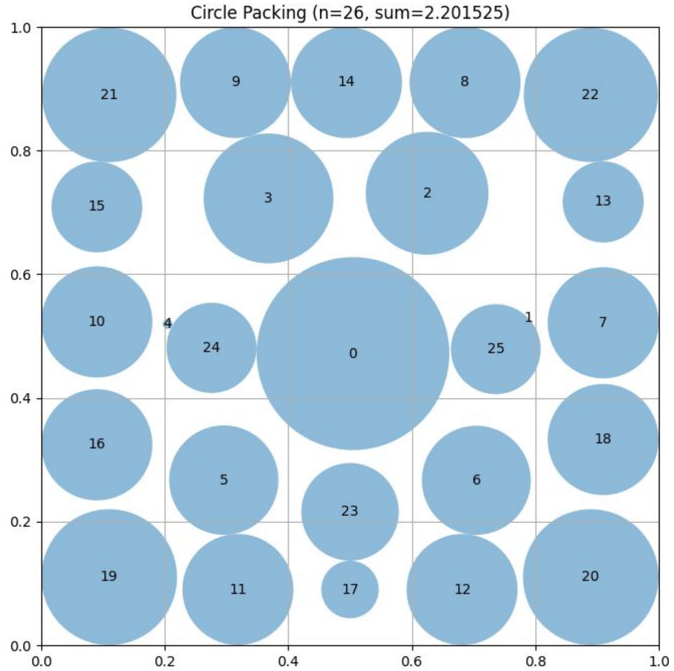


```
# Generation 10
# Parameters for the arrangement (fine-tuned)
r_center = 0.1675  # Central circle radius

# 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
```

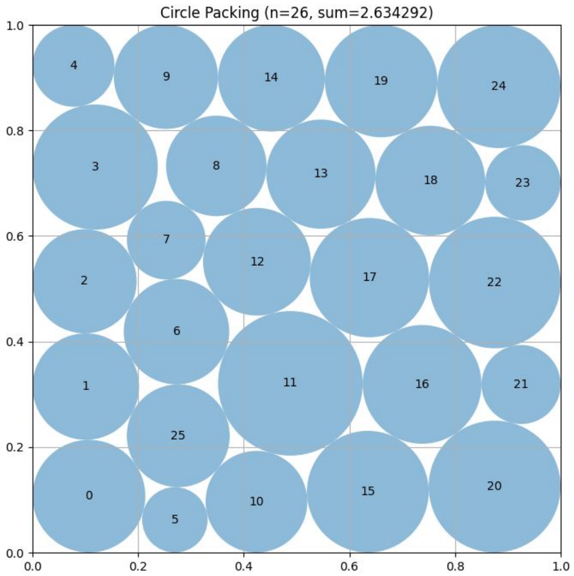


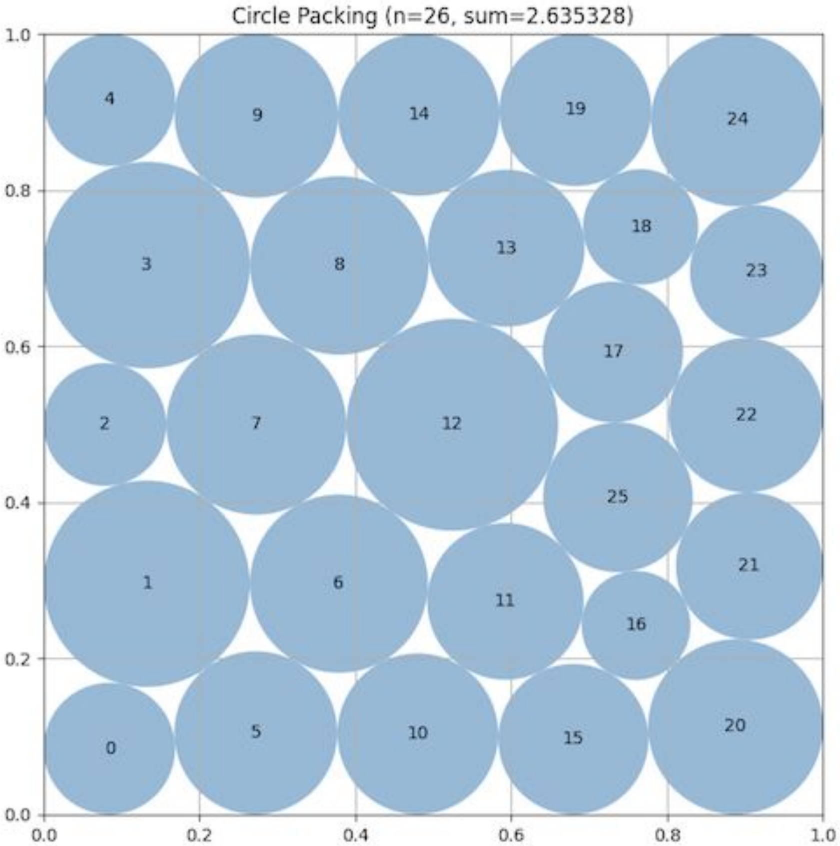
After 200+ iterations

```
# 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
```

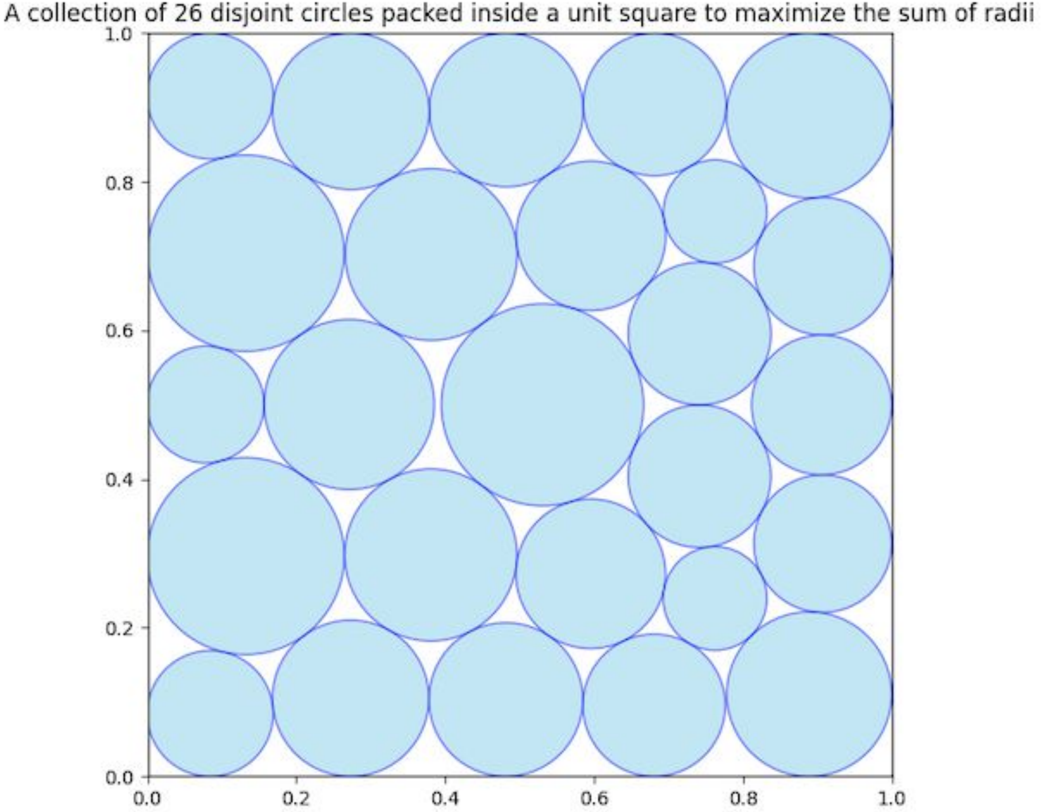


```python
# Final solution with scipy.optimize
def construct_packing():
    # ... initialization code ...

    # 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 the 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)
```





OpenEvolve



AlphaEvolve

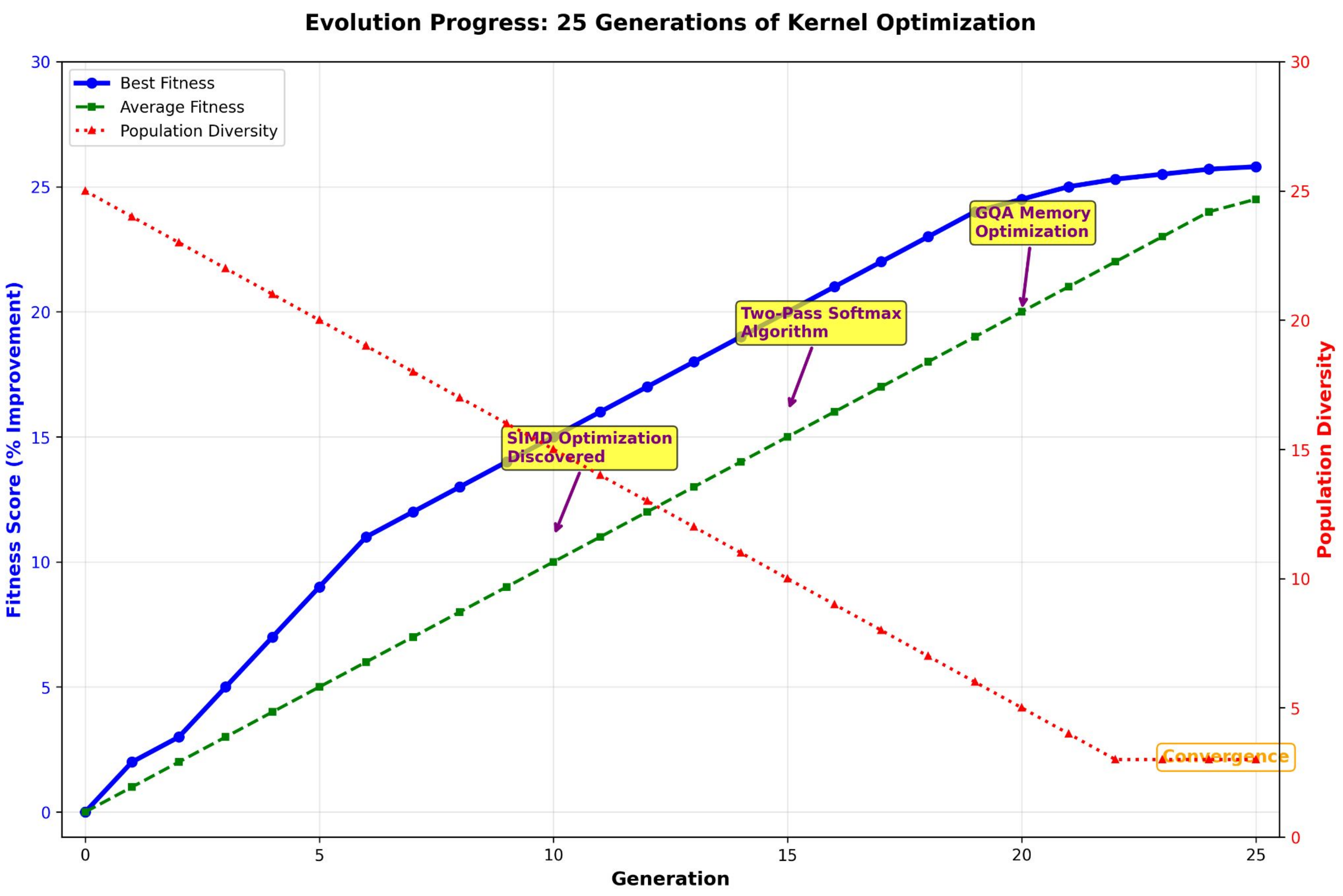
MLX Metal Kernel for Transformer Attention

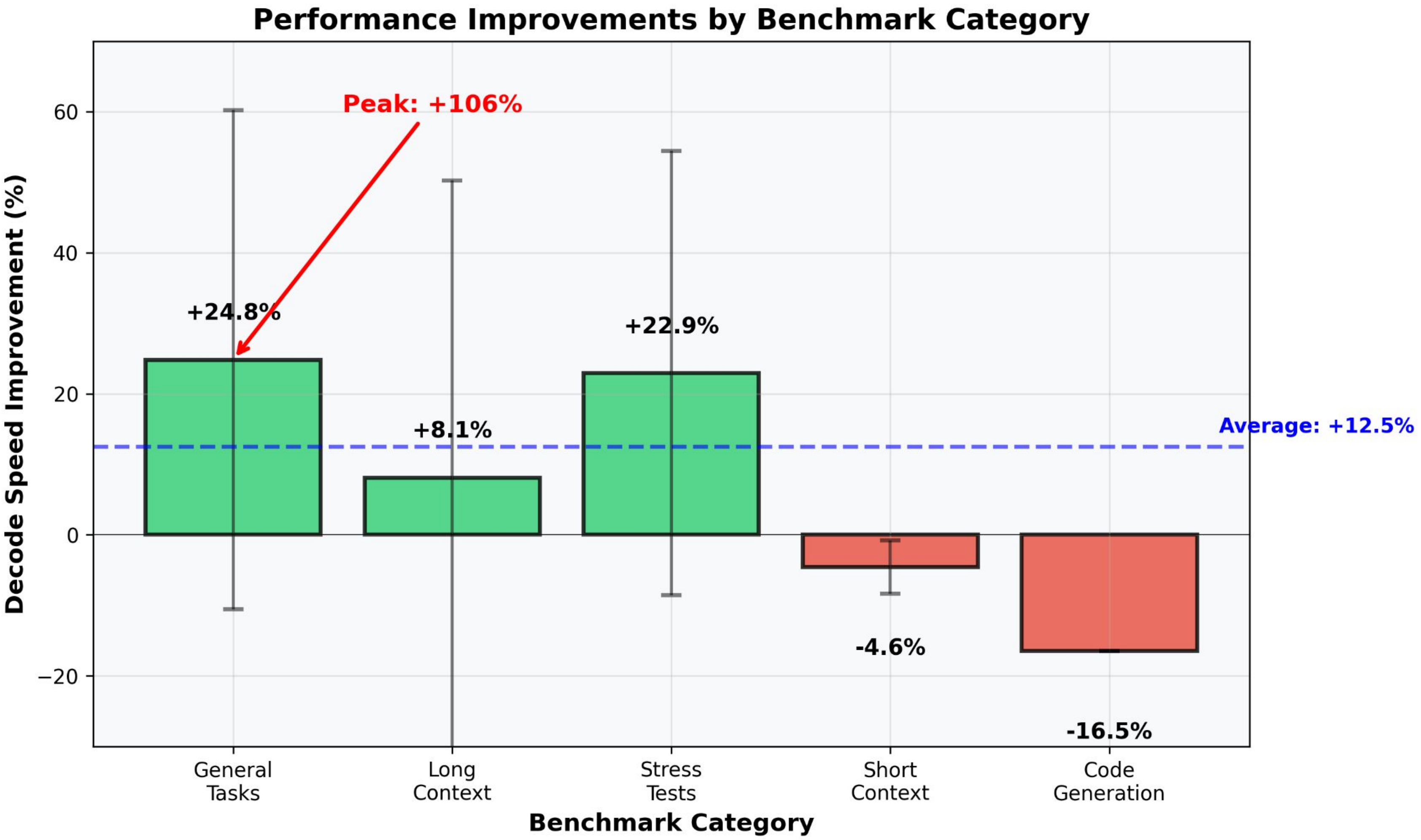
The Challenge

- **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
Fitness Score (% Improvement)
Population Diversity
Generation
SIMD Optimization Discovered
Two-Pass Softmax Algorithm
GQA Memory Optimization
Convergence
0
5
10
15
20
25
30



Performance Improvements by Benchmark Category
Peak: +106%
+24.8%
+8.1%
+22.9%
-4.6%
-16.5%
Average: +12.5%
60
40
20
0
−20
Decode Speed Improvement (%)
General Tasks
Long Context
Stress Tests
Short Context
Code Generation
Benchmark Category

AlgoTune

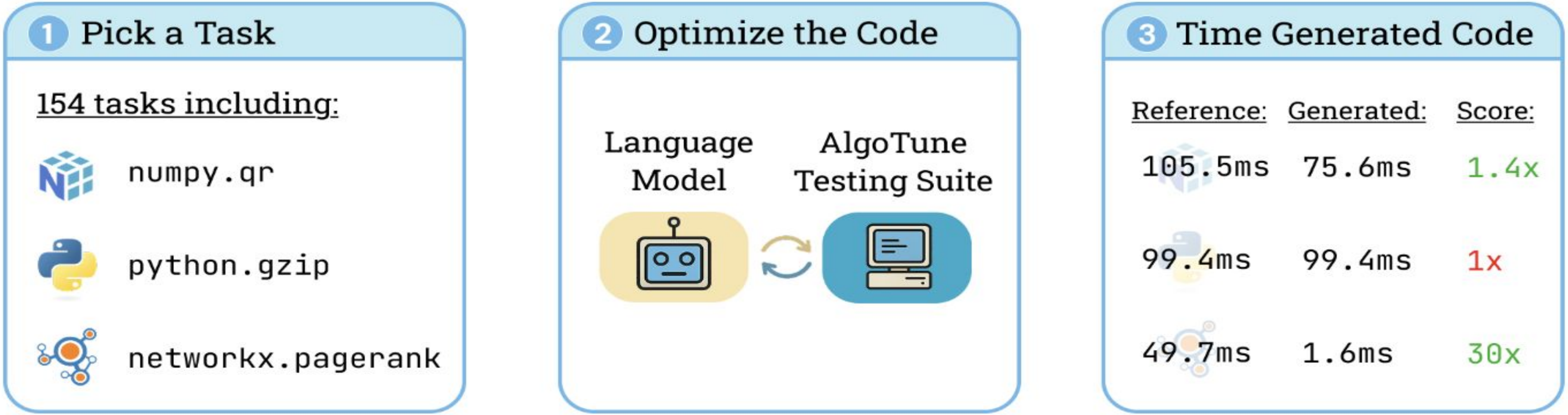
Can Language Models Speed Up General-Purpose Numerical Programs?

UNIVERSITÄT TÜBINGEN

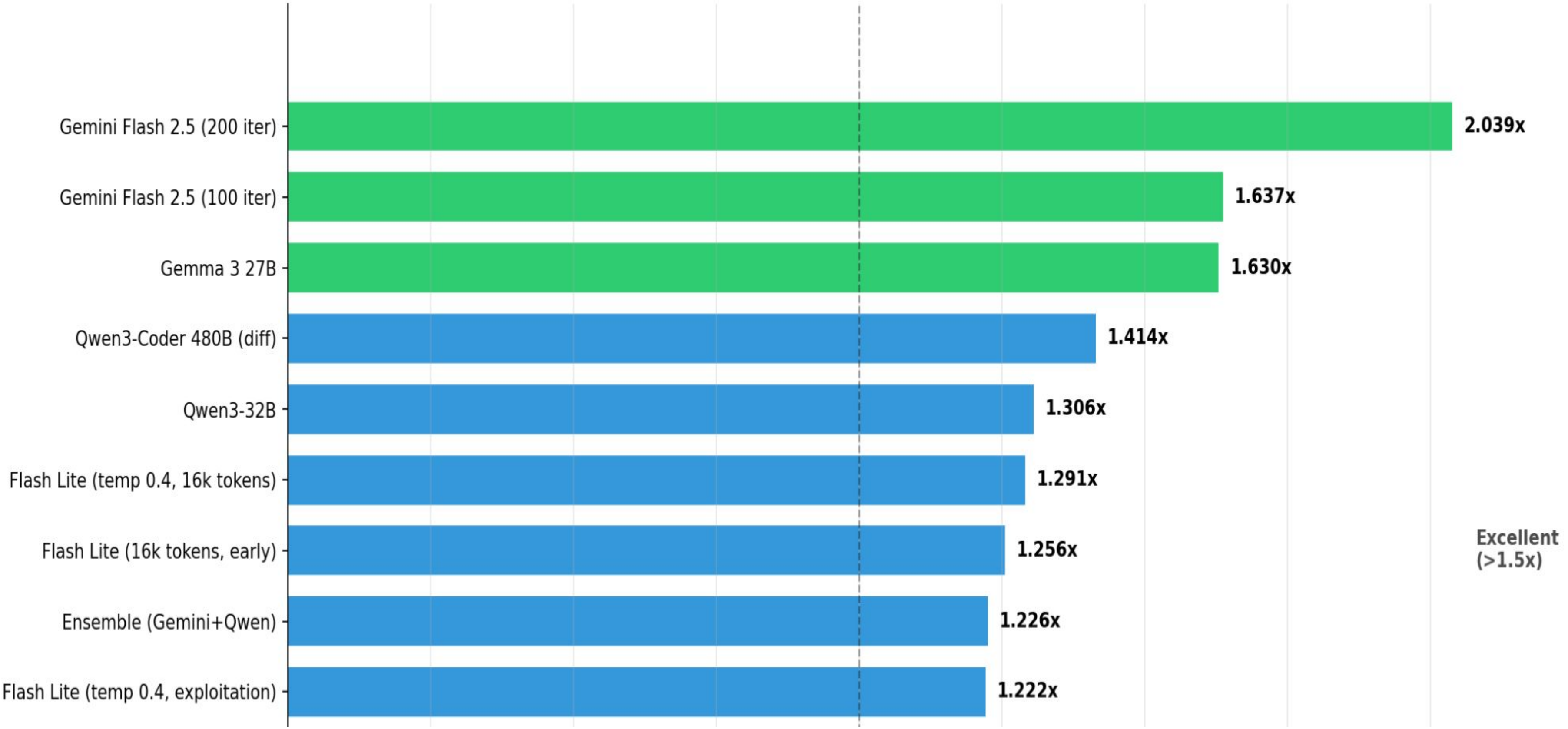
Ori Press Brandon Amos Haoyu Zhao Yikai Wu Samuel K. Ainsworth Dominik Krupke Patrick Kidger
Touqir Sajed Bartolomeo Stellato Jisun Park Nathanael Bosch Eli Meril Albert Steppi
Arman Zharmagambetov Fangzhao Zhang David Pérez-Piñeiro Alberto Mercurio Ni Zhan
Talor Abramovich Kilian Lieret Hanlin Zhang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

PRINCETON UNIVERSITY

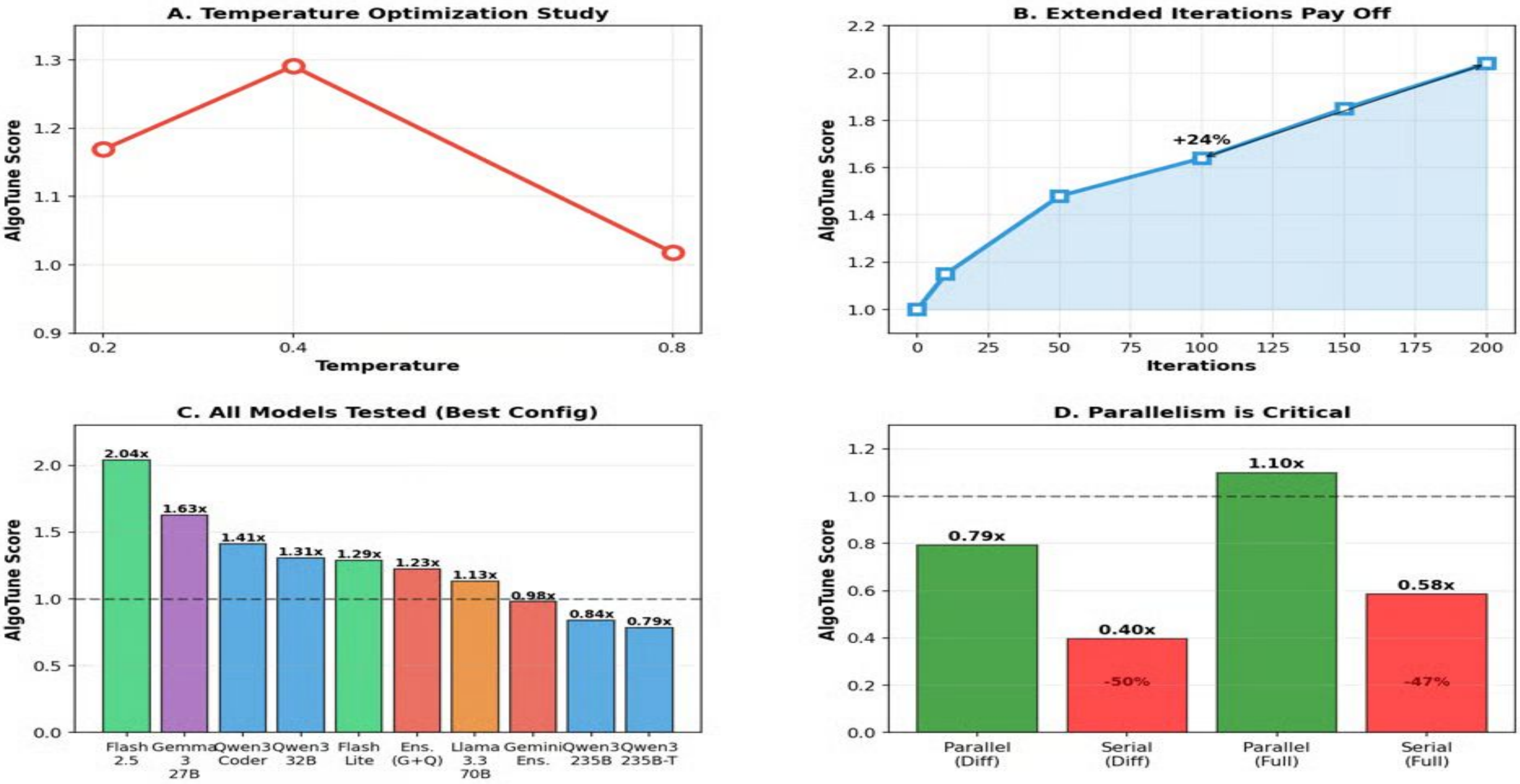
Paper   Code

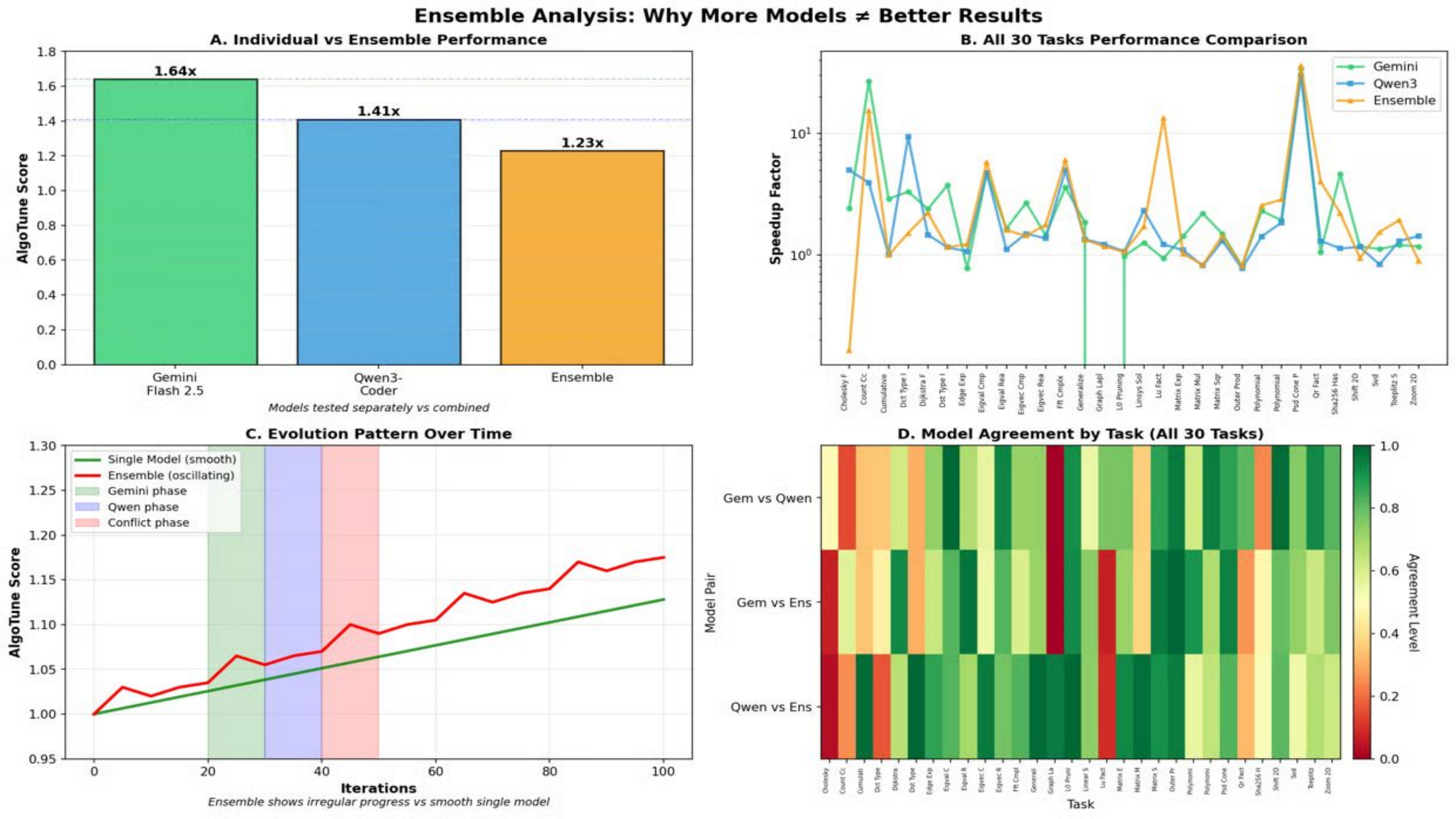


OpenEvolve + AlgoTune: Complete Experimental Results (All 29 Configurations)



AlgoTune Experiments: Executive Summary





Ensemble Analysis: Why More Models ≠ Better Results
A. Individual vs Ensemble Performance
AlgoTune Score
1.64x
1.41x
1.23x
Gemini Flash 2.5
Qwen3-Coder
Ensemble
Models tested separately vs combined
B. All 30 Tasks Performance Comparison
Speedup Factor
Gemini
Qwen3
Ensemble
Cholesky F
Count Cc
Cumulative
Dct Type I
Dijkstra F
Dst Type I
Edge Exp
Eigval Cmp
Eigval Rea
Eigvec Cmp
Eigvec Rea
Fft Cmplx
Generalize
Graph Lapl
L0 Pruning
Linsys Sol
Lu Fact
Matrix Exp
Matrix Mul
Matrix Sqr
Outer Prod
Polynomial
Polynomial
Psd Cone P
Qr Fact
Sha256 Has
Shift 2D
Svd
Toeplitz S
Zoom 2D
C. Evolution Pattern Over Time
Single Model (smooth)
Ensemble (oscillating)
Gemini phase
Qwen phase
Conflict phase
AlgoTune Score
Iterations
Ensemble shows irregular progress vs smooth single model
D. Model Agreement by Task (All 30 Tasks)
Model Pair
Gem vs Qwen
Gem vs Ens
Qwen vs Ens
Task
Agreement Level

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