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]

Slides created by Yegon Kim
2025/10/01

https://arxiv.org/abs/2506.13131

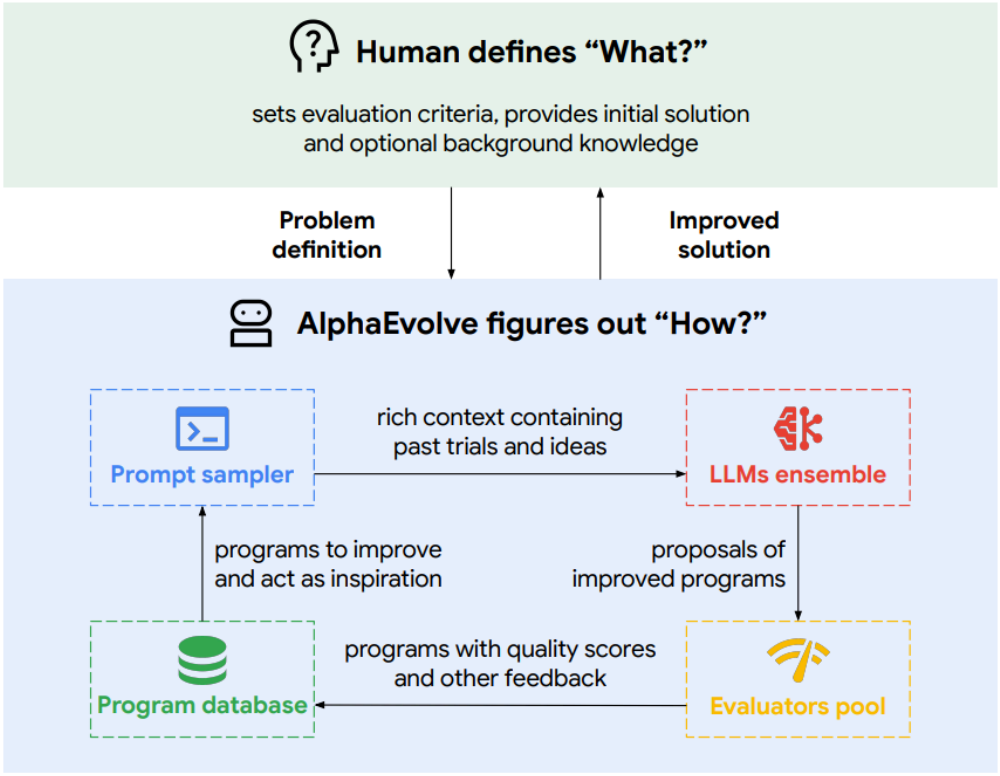
Problem Setting

- Some problems are “easy to evaluate” but “hard to solve”
- We focus on code generation, which can be automatically evaluated

Motivation

- LLMs can generating promising ideas (although they are often wrong)
- They can also improve ideas when given feedback
- We exploit these features using an *evolutionary* approach, where the LLM serves as the *mutation* operation

Overview


Human defines “What?”
sets evaluation criteria, provides initial solution and optional background knowledge
Problem definition
Improved solution
AlphaEvolve figures out “How?”
Prompt sampler
rich context containing past trials and ideas
LLMs ensemble
programs to improve and act as inspiration
proposals of improved programs
Program database
programs with quality scores and other feedback
Evaluators pool

Task Specification

- Evaluation function `evaluate(h)`
- `# EVOLVE-BLOCK-START` and `# EVOLVE-BLOCK-END`

```python
# EVOLVE-BLOCK START
"""Image classification experiment in jaxline."""

import jax
...
# EVOLVE-BLOCK-END

...

# EVOLVE-BLOCK-START
class ConvNet(hk.Module):
  def __init__(self, num_classes): ...
  def __call__(self, inputs, is_training): ...


def sweep():
  return hyper.zipit([...])
# EVOLVE-BLOCK-END

...

def evaluate(eval_inputs) -> dict[str, float]:
  ...
  return metrics
```

- Two choices are possible

1) AlphaEvolve → What we want

2) AlphaEvolve → Search algorithm → What we want

Prompt

- Explicit context
- Stochastic formatting (for diversity)
- Evaluation results
- Meta prompt (evolved by LLM in another step)

```
Act as an expert software developer. Your task is to iteratively
improve the provided codebase. [...]

- Prior programs

Previously we found that the following programs performed well
on the task at hand:

top_1_acc: 0.796; neg_eval_log_loss: 0.230; average_score: 0.513

"""Image classification experiment in jaxline."""
[...]
class ConvNet(hk.Module):
  """Network."""

  def __init__(self, num_channels=32, num_output_classess=10):
    super().__init__()
    self._conv1 = hk.Conv2D(num_channels, kernel_shape=3)
    self._conv2 = hk.Conv2D(num_channels * 2, kernel_shape=3)
    self._conv3 = hk.Conv2D(num_channels * 4, kernel_shape=3)
    self._logits_module = hk.Linear(num_output_classes)
[...]


- Current program

Here is the current program we are trying to improve (you will
need to propose a modification to it below).

top_1_acc: 0.862; neg_eval_log_loss: 0.387; average_score: 0.624

"""Image classification experiment in jaxline."""
[...]
class ConvNet(hk.Module):
  """Network."""

  def __init__(self, num_channels=32, num_output_classes=10):
    super().__init__()
    self._conv1 = hk.Conv2D(num_channels, kernel_shape=3)
    self._block1 = ResNetBlock(num_channels)
    self._block2 = ResNetBlock(num_channels * 2, stride=2)
    self._block3 = ResNetBlock(num_channels * 4, stride=2)
    self._logits_module = hk.Linear(num_output_classes)
[...]

SEARCH/REPLACE block rules:
[...]

Make sure that the changes you propose are consistent with each
other. For example, if you refer to a new config variable
somewhere, you should also propose a change to add that
variable.

Example:
[...]

Task
Suggest a new idea to improve the code that is inspired by your
expert knowledge of optimization and machine learning.

Describe each change with a SEARCH/REPLACE block.
```

Generation

- `SEARCH` and `REPLACE` blocks
- Uses both Gemini 2.0 Flash and Gemini 2.0 Pro

```
The current model uses a simple ResNet architecture with only
three ResNet blocks. We can improve its performance by
increasing the model capacity and adding regularization. This
will allow the model to learn more complex features and
generalize better to unseen data. We also add weight decay to
the optimizer to further regularize the model and prevent
overfitting. AdamW is generally a better choice than Adam,
especially with weight decay.


<<<<<<< SEARCH
    self._block1 = ResNetBlock(num_channels)
    self._block2 = ResNetBlock(num_channels * 2, stride=2)
    self._block3 = ResNetBlock(num_channels * 4, stride=2)
=======
    self._block1 = ResNetBlock(num_channels)
    self._block2 = ResNetBlock(num_channels, stride=1)
    self._block3 = ResNetBlock(num_channels * 2, stride=2)
    self._block4 = ResNetBlock(num_channels * 2, stride=1)
    self._block5 = ResNetBlock(num_channels * 4, stride=2)
    self._block6 = ResNetBlock(num_channels * 4, stride=1)
>>>>>>> REPLACE
<<<<<<< SEARCH
  def optimizer(self, learning_rate):
    return optax.adam(learning_rate)
=======
  def optimizer(self, learning_rate):
    return optax.adamw(learning_rate, weight_decay=1e-4)
>>>>>>> REPLACE
```

Evaluation

- `evaluate`
- One evaluation run can take hours. AlphaEvolve's main contribution is in sample efficiency.
- Evaluation cascade with easier test cases first
- Parallelized computation
- Multiple scores — also provided instrumental benefit

Evolutionary Algorithm

- Database of (program, score) pairs
- Evolutionary algorithm inspired by MAP elites and island-based population models
- Asynchronous computational pipeline

Results

- matrix multiplication (improves upon SOTA in 14 cases)
- notably, the first breakthrough in 4x4 after Strassen (1969)
- AlphaEvolve outputs search algorithm (Adam optimizer based) that finds short matmul algorithm

$\langle m, n, p \rangle$	best known [reference]	*AlphaEvolve*
$\langle 2, 4, 5 \rangle$	33 [42]	**32**
$\langle 2, 4, 7 \rangle$	46 [93]	**45**
$\langle 2, 4, 8 \rangle$	52 [93]	**51**
$\langle 2, 5, 6 \rangle$	48 [93]	**47**
$\langle 3, 3, 3 \rangle$	23 [52]	23
$\langle 3, 4, 6 \rangle$	56 [48]	**54**
$\langle 3, 4, 7 \rangle$	66 [91]	**63**
$\langle 3, 4, 8 \rangle$	75 [91]	**74**
$\langle 3, 5, 6 \rangle$	70 [48]	**68**
$\langle 3, 5, 7 \rangle$	82 [91]	**80**
$\langle 4, 4, 4 \rangle$	49 [95]	**48**
$\langle 4, 4, 5 \rangle$	62 [47]	**61**
$\langle 4, 4, 7 \rangle$	87 [93]	**85**
$\langle 4, 4, 8 \rangle$	98 [95]	**96**
$\langle 4, 5, 6 \rangle$	93 [48]	**90**
$\langle 5, 5, 5 \rangle$	93 [72]	93

Matrix multiplication

Results

- >50 problems in mathematic that involve construction of objects
- matches SOTA in ~75%, improves upon SOTA in ~25%

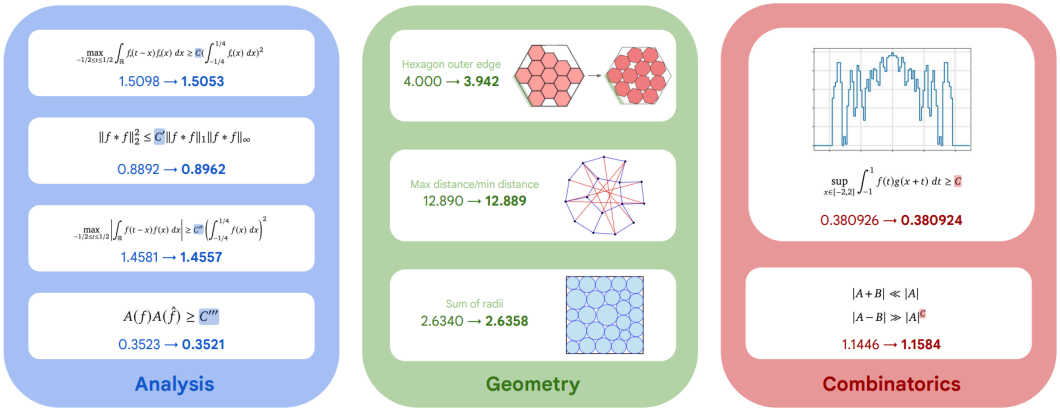


Figure 5 | Examples of SOTA-breaking mathematical constructions discovered with *AlphaEvolve*. The versatility of *AlphaEvolve* allows us to tackle problems in analysis (autocorrelation and uncertainty inequalities), geometry (packing and minimum/maximum distance problems) and combinatorics (Erdős's minimum overlap problem and sums and differences of finite sets).

Results

- job scheduling heuristics
- tiling strategy in matmul kernels
- arithmetic circuits used within TPUs
- compiler-generated code for self-attention

- all results were adopted into Google's computing infrastructure

How I got the highest score on ARC-AGI again swapping Python for English

Using Multi-Agent Collaboration with Evolutionary Test-Time Compute

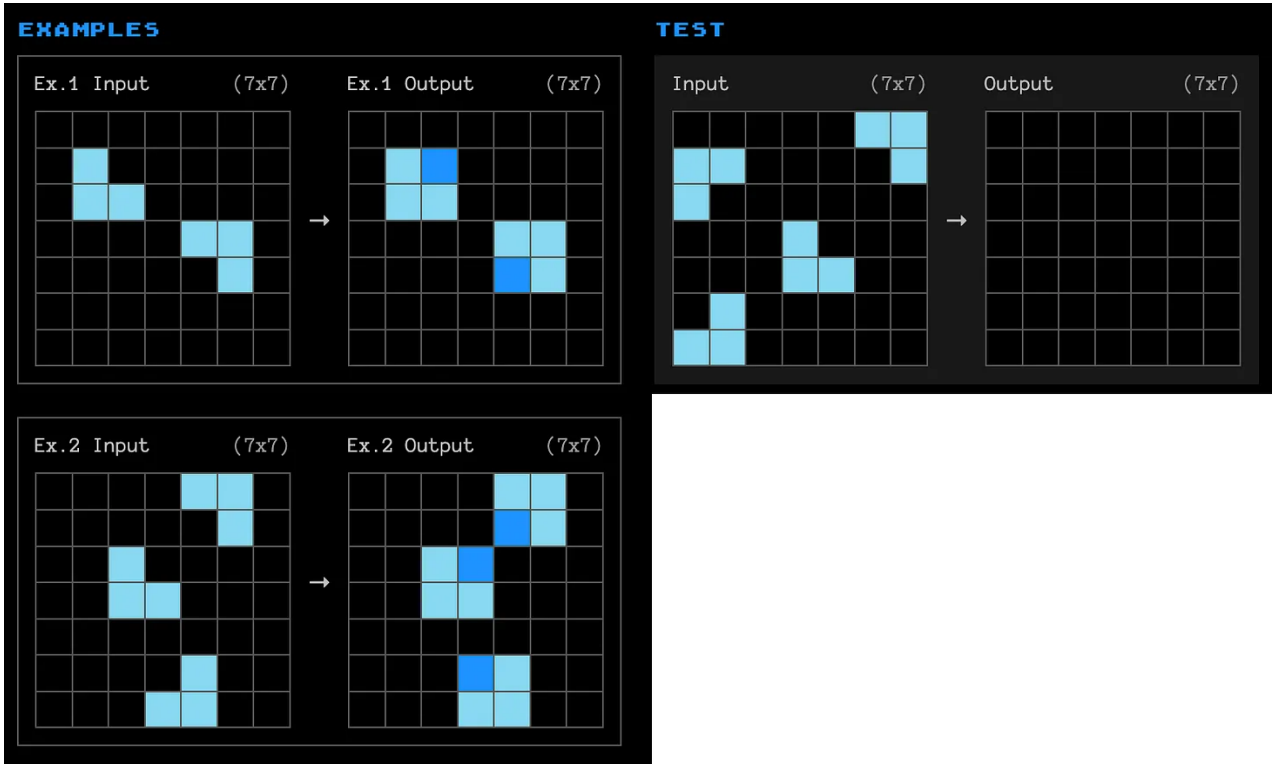
JEREMY BERMAN

SEP 17, 2025

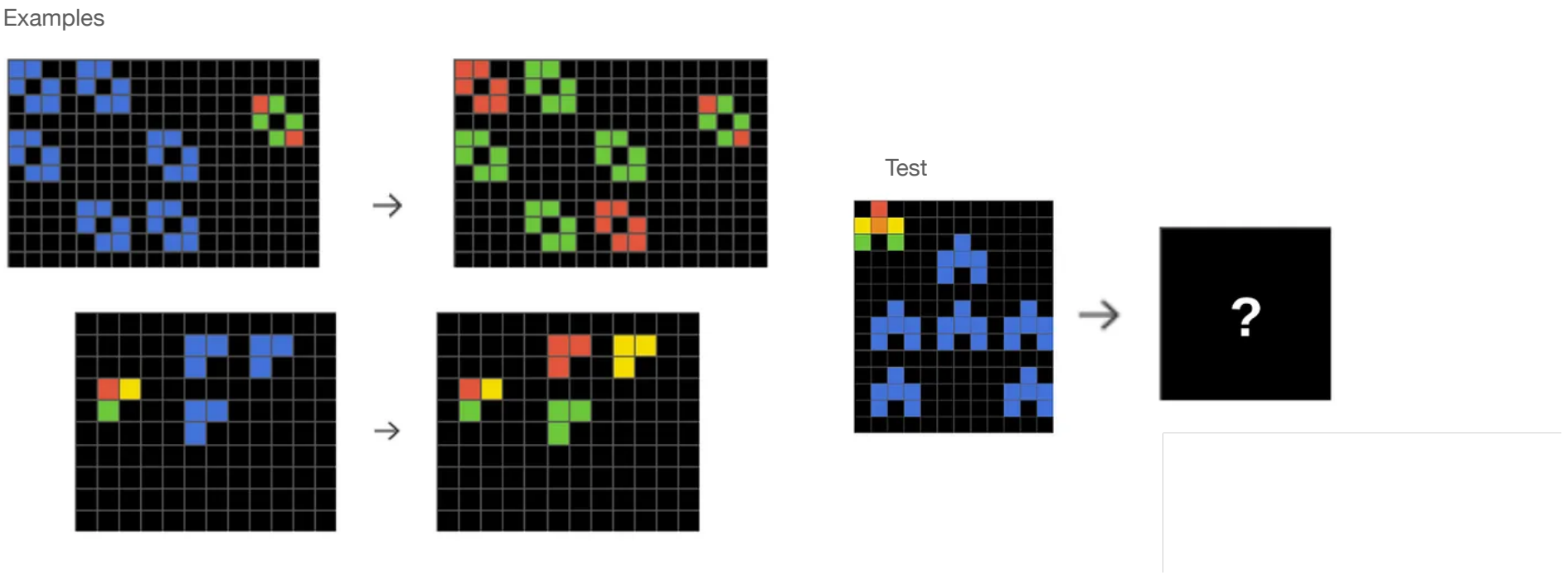
I think ARC-AGI is *still* the most important benchmark we have today. It's surprising that LLMs can win the math olympiad but struggle with simple puzzles that humans can solve easily.

This highlights a core limitation of current LLMs: they struggle to reason about things they weren't trained on. They struggle to generalize. But they are getting better, fast.

ARC-AGI

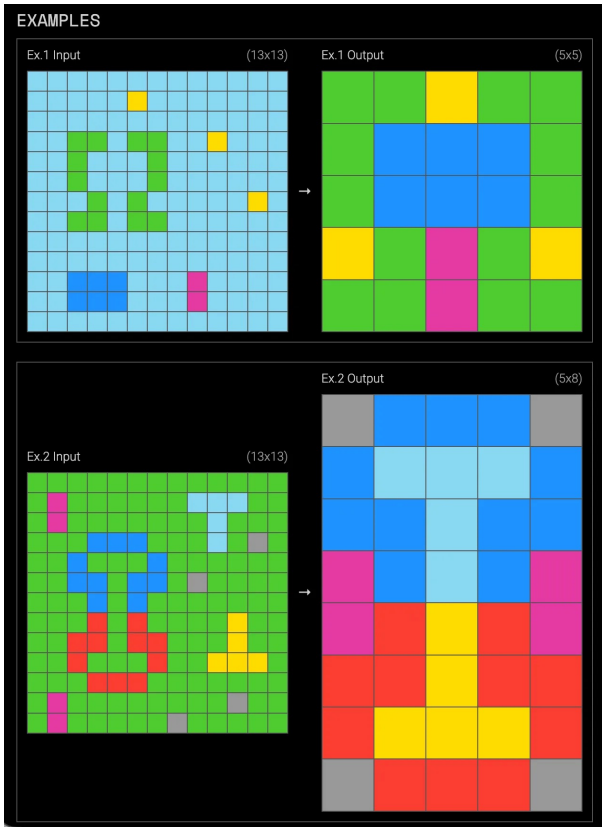

EXAMPLES
Ex.1 Input (7x7)
Ex.1 Output (7x7)
Ex.2 Input (7x7)
Ex.2 Output (7x7)
TEST
Input (7x7)
Output (7x7)

ARC-AGI

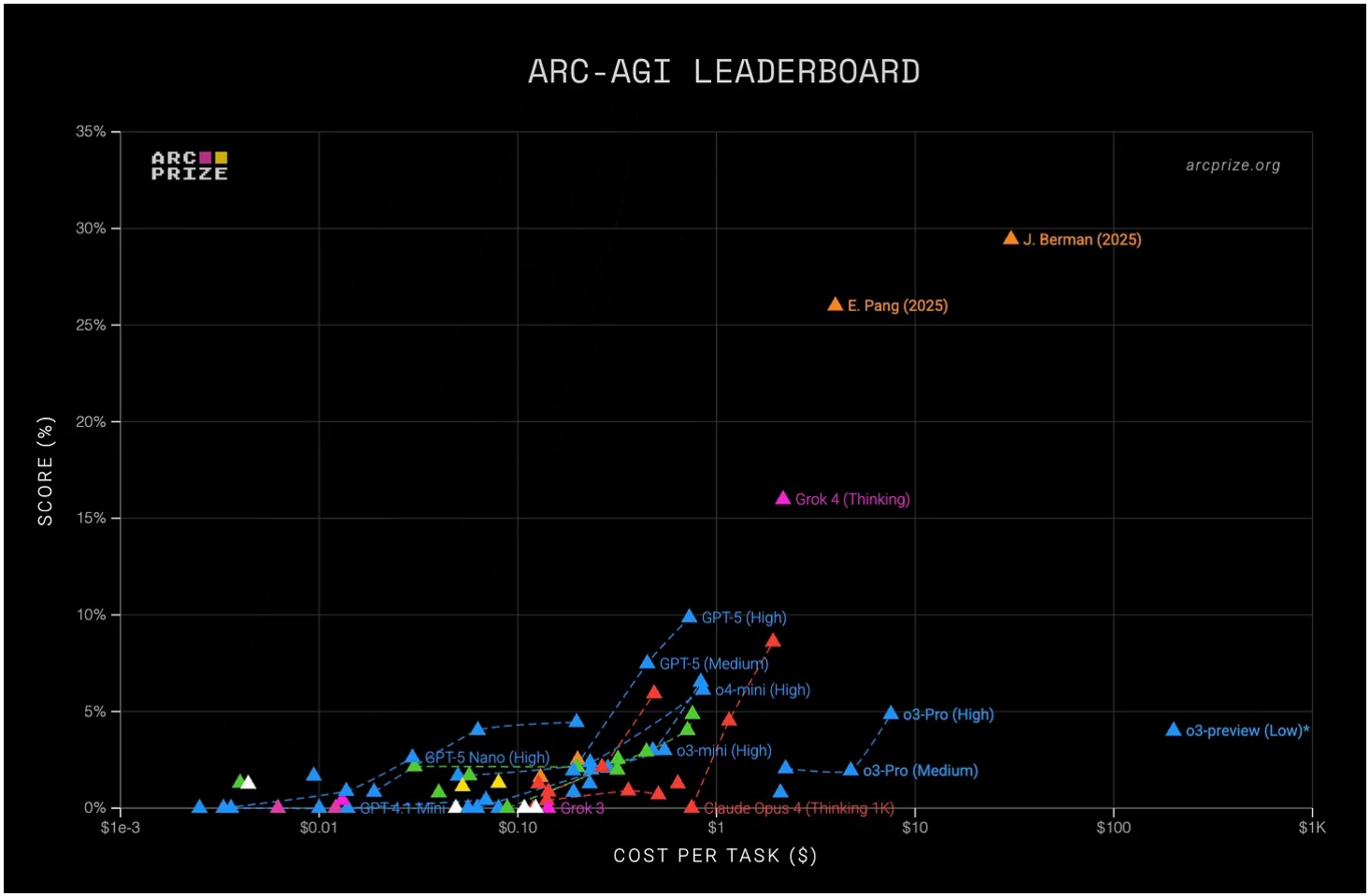


ARC-AGI

- After o3 got >75% on ARC-AGI v1, Chollet and his team published ARC-AGI v2
- Compared to ARC-AGI v1, the transformation rules are more complicated
- Grand Prize ($700K) for >85% on v2

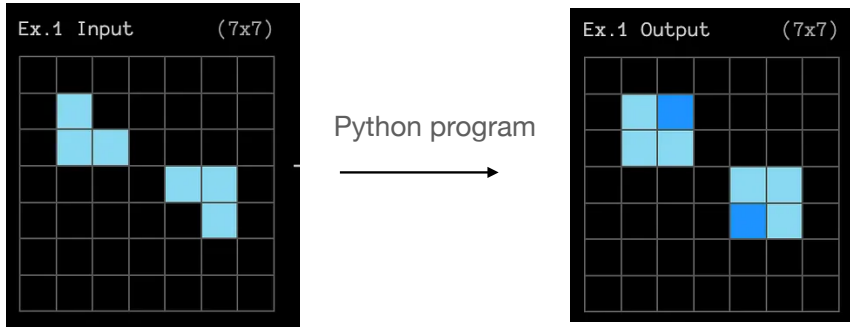


v2 Progress


ARC-AGI LEADERBOARD
ARC PRIZE
arcprize.org
SCORE (%)
35%
30%
25%
20%
15%
10%
5%
0%
$1e-3
$0.01
$0.10
$1
$10
$100
$1K
COST PER TASK ($)
J. Berman (2025)
E. Pang (2025)
Grok 4 (Thinking)
GPT-5 (High)
GPT-5 (Medium)
o4-mini (High)
o3-Pro (High)
o3-preview (Low)*
o3-mini (High)
GPT-5 Nano (High)
o3-Pro (Medium)
GPT-4.1 Mini
Grok 3
Claude Opus 4 (Thinking 1K)

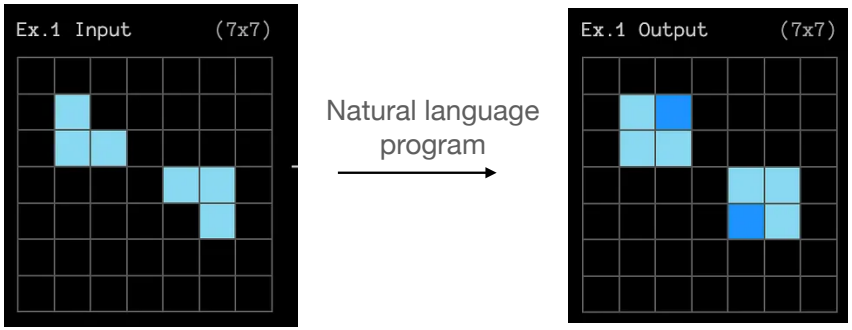
Method

- 2024 December SOTA on v1 was "Evolutionary Test-Time Compute" by Berner
- Program synthesis approach
- Evolution of Python programs with LLM as mutation
- Evaluation and feedback with grid-wise and pixel-wise diff



Method

- Berner found that v2's transformations are too complex to express simply with Python programs
- Uses *natural language* instead of Python
- Evaluation and Inference is done by transforming input grid with LLM and natural language instruction
  i.e. LLM is the "code interpreter"



References

- https://arxiv.org/abs/2506.13131
- https://www.jasonwei.net/blog/asymmetry-of-verification-and-verifiers-law
- https://jeremyberman.substack.com/p/how-i-got-the-highest-score-on-arc-agi-again

Other Recent Works

- "GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning" (Agrawal et al. 2025)
- "Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents" (Zhang et al. 2025)
- "ShinkaEvolve: Towards Open-Ended And Sample-Efficient Program Evolution" (Lange et al. 2025)