AlphaEvolve: A coding agent for scientific and algorithmic discovery

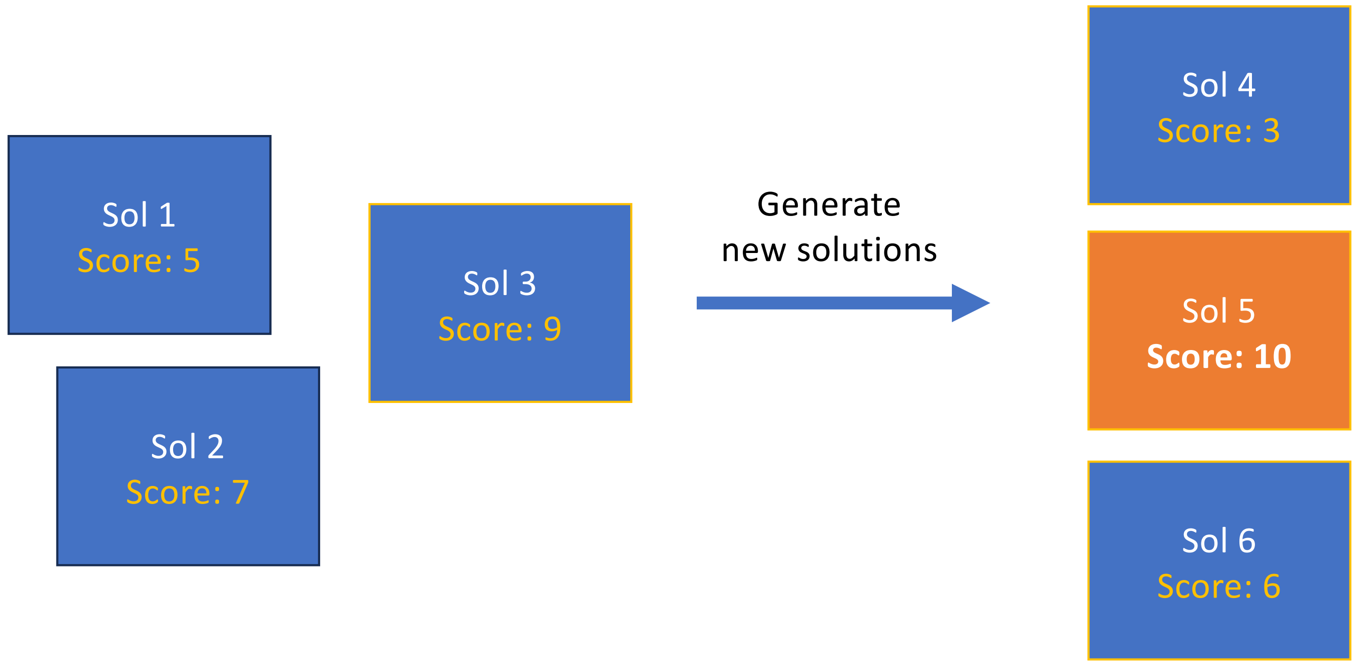
Alexander Novikov*, **Ngân Vũ***, **Marvin Eisenberger***, **Emilien Dupont***, **Po-Sen Huang***, **Adam Zsolt Wagner***, **Sergey Shirobokov***, **Borislav Kozlovskii***, **Francisco J. R. Ruiz**, **Abbas Mehrabian**, **M. Pawan Kumar**, **Abigail See**, **Swarat Chaudhuri**, **George Holland**, **Alex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

Presented by:

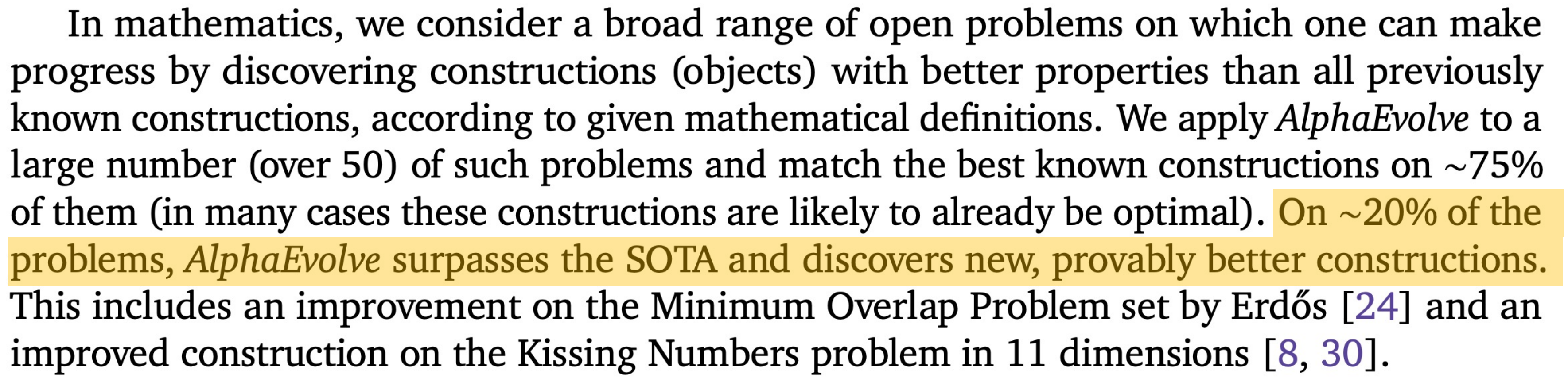
John Tan Chong Min

Question

- Can we evolve **solutions** to score better on an **evaluation function**?
- Insights: Natural selection scoring pressure + LLM-based code evolution



AlphaEvolve: Impressive Performance


In mathematics, we consider a broad range of open problems on which one can make progress by discovering constructions (objects) with better properties than all previously known constructions, according to given mathematical definitions. We apply *AlphaEvolve* to a large number (over 50) of such problems and match the best known constructions on ~75% of them (in many cases these constructions are likely to already be optimal). On ~20% of the problems, *AlphaEvolve* surpasses the SOTA and discovers new, provably better constructions. This includes an improvement on the Minimum Overlap Problem set by Erdős [24] and an improved construction on the Kissing Numbers problem in 11 dimensions [8, 30].

Human-in-the-loop self-improvement

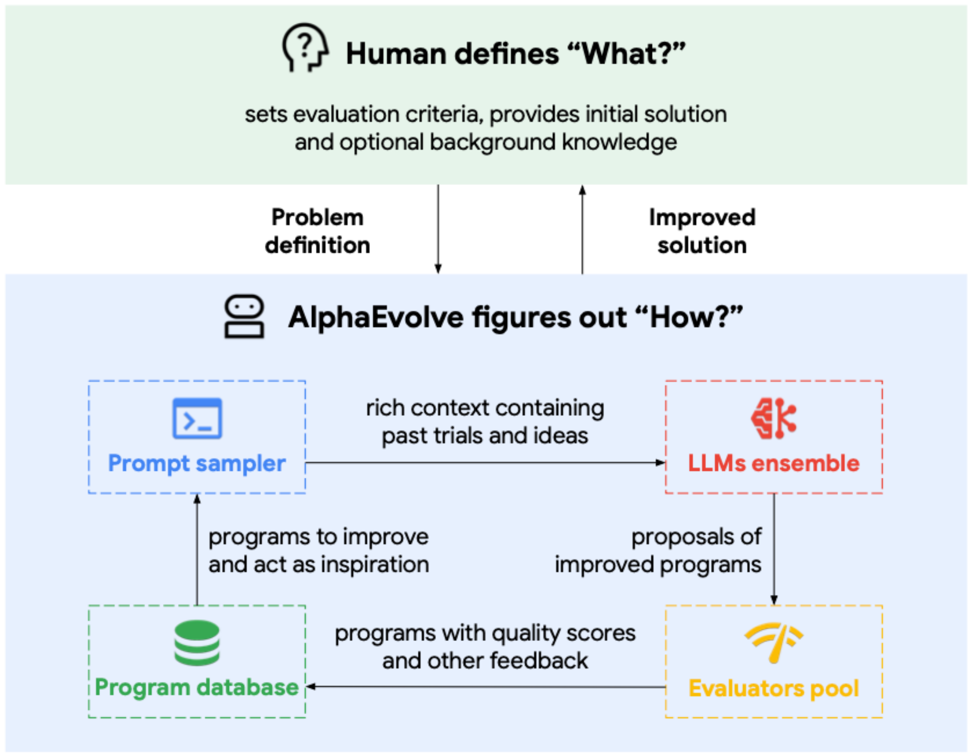
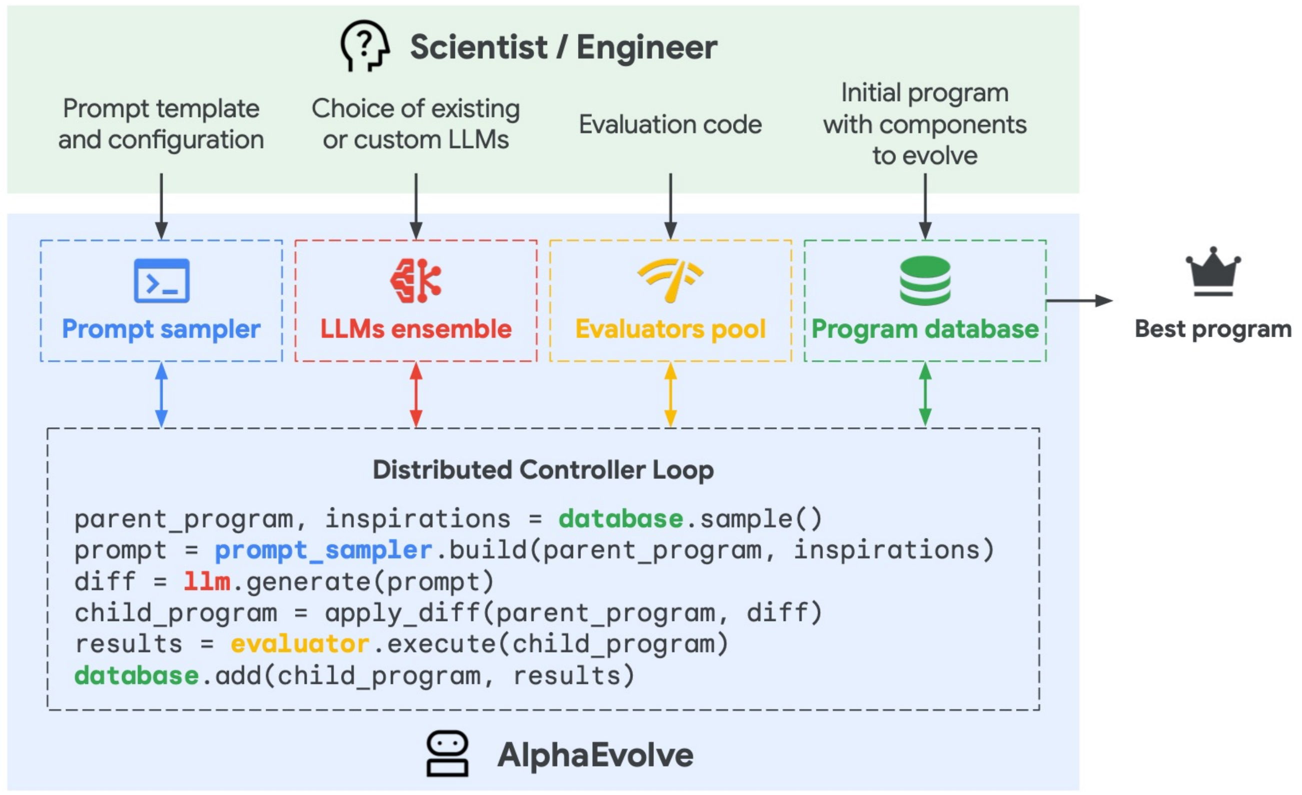

Human defines "What?"
sets evaluation criteria, provides initial solution and optional background knowledge
Problem definition
Improved solution
AlphaEvolve figures out "How?"
Prompt sampler
rich context containing past trials and ideas
LLMs ensemble
programs to improve and act as inspiration
proposals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

Figure 1 | *AlphaEvolve* high-level overview.

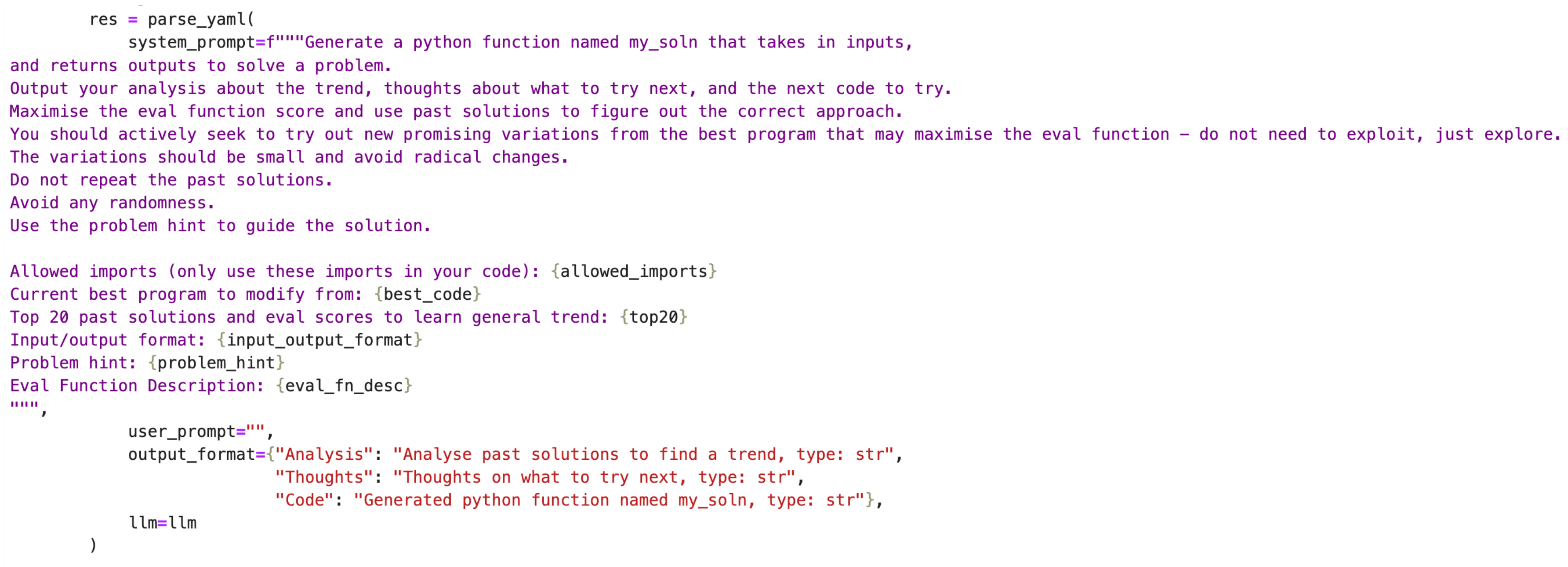
Each step of process can be augmented by humans


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
AlphaEvolve

```
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)
```

Demo

- Finding out whether an evolution system can learn my name (john)



```
        res = parse_yaml(
            system_prompt=f"""Generate a python function named my_soln that takes in inputs,
and returns outputs to solve a problem.
Output your analysis about the trend, thoughts about what to try next, and the next code to try.
Maximise the eval function score and use past solutions to figure out the correct approach.
You should actively seek to try out new promising variations from the best program that may maximise the eval function - do not need to exploit, just explore.
The variations should be small and avoid radical changes.
Do not repeat the past solutions.
Avoid any randomness.
Use the problem hint to guide the solution.

Allowed imports (only use these imports in your code): {allowed_imports}
Current best program to modify from: {best_code}
Top 20 past solutions and eval scores to learn general trend: {top20}
Input/output format: {input_output_format}
Problem hint: {problem_hint}
Eval Function Description: {eval_fn_desc}
""",
            user_prompt="",
            output_format={"Analysis": "Analyse past solutions to find a trend, type: str",
                           "Thoughts": "Thoughts on what to try next, type: str",
                           "Code": "Generated python function named my_soln, type: str"},
            llm=llm
        )
```

https://github.com/tanchongmin/agentjo/blob/main/contrib/Fun_AgentJo_Projects/AlphaEvolve.ipynb

Demo Results

Overview

- Tries to guess the word john, given the input-output description, eval function description and a hint

- Evaluation ranges from ordinal difference between characters of guessed word and ground truth, to absolute score of number of correct letters

- LLM actually can perform better with absolute score, as it tells with certainty whether a letter is correct - smooth continuous eval functions may not be the best

https://github.com/tanchongmin/agentjo/blob/main/contrib/Fun_AgentJo_Projects/AlphaEvolve.ipynb

Demo Results

Hints

- Tried with various hints
- Easy: "You are to output john"
- Medium: "You are to generate and output a 4-letter English word starting with j"
- Hard: "You are to generate and output a 4-letter English word"

Results

- Progress is made for all, solving Easy and Medium (Easy solved in 1 try for either eval, Medium solved in 29 tries for continuous eval, 12 tries for absolute eval)
- Hard is difficult as search space is large, and LLMs do not handle letters well.
- Also, lack of stochasticity may lead to sampling same solution again as the worse solution may not appear in the prompt
- Also, tried with adversarial hints, if hint is outright wrong (e.g. output mary), it does not work.
- If hint is wrong but towards solution (output a girl's name), it can work.

https://github.com/tanchongmin/agentjo/blob/main/contrib/Fun_AgentJo_Projects/AlphaEvolve.ipynb

Motivation

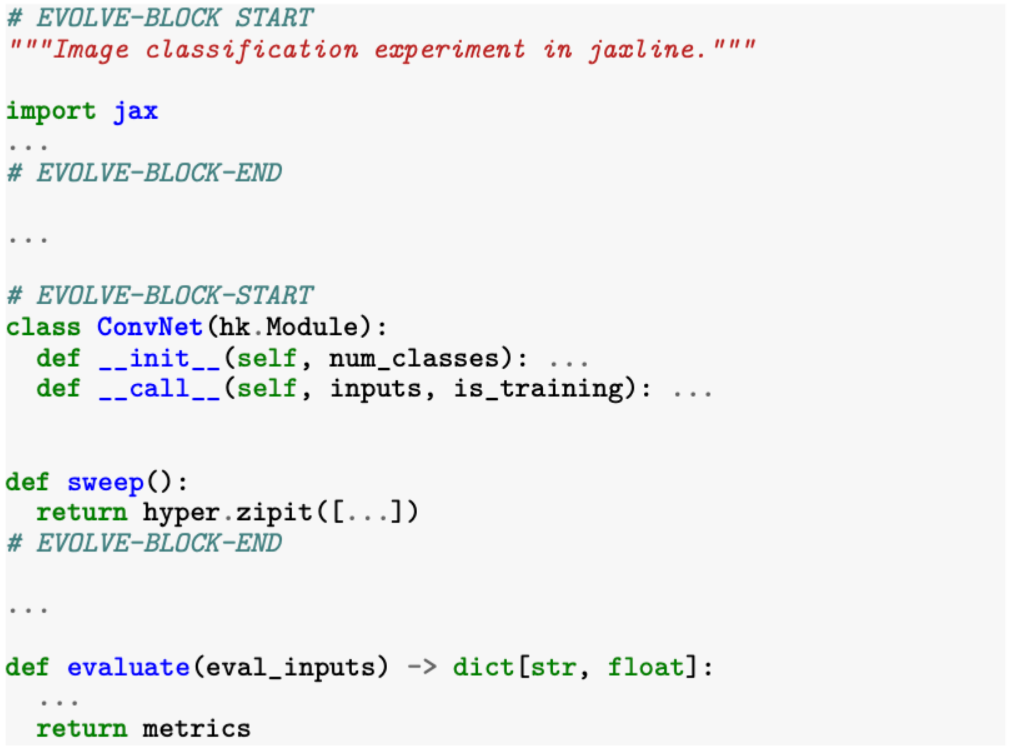
- AlphaEvolve represents the candidates (for example, new mathematical objects or practical heuristics) as algorithms and uses a set of LLMs to generate, critique, and evolve a pool of such algorithms

- The LLM-directed evolution process is **grounded using code execution** and **automatic evaluation**

- This evaluation mechanism allows AlphaEvolve to avoid any incorrect suggestions from the base LLM

Step through the process

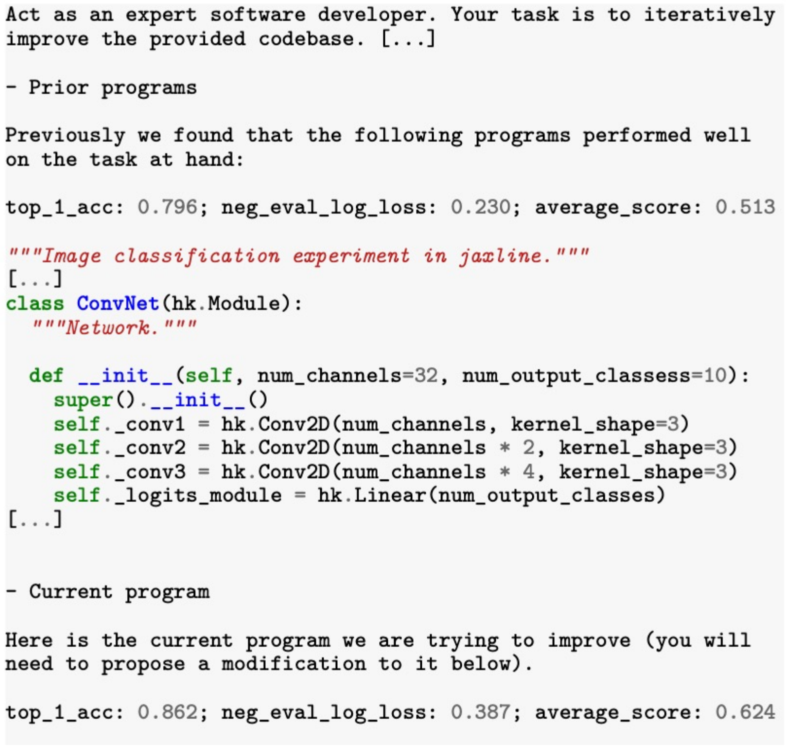
Step 1: Provide blocks of code to evolve



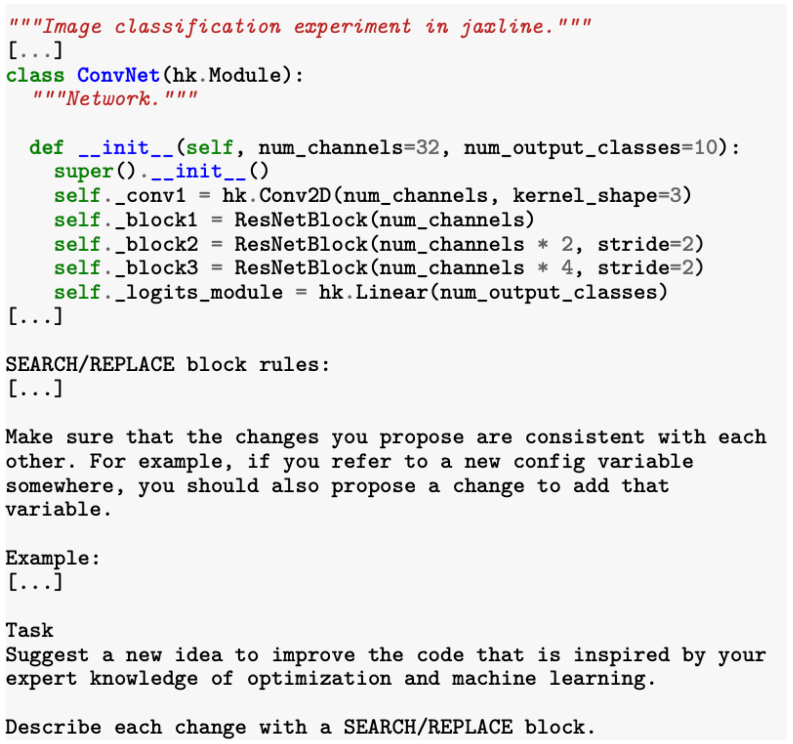
```
# EVOLVE-BLOCK START
"""Image classification experiment in jaxline."""

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
```

(a)

Step 2: Prompt to generate new code



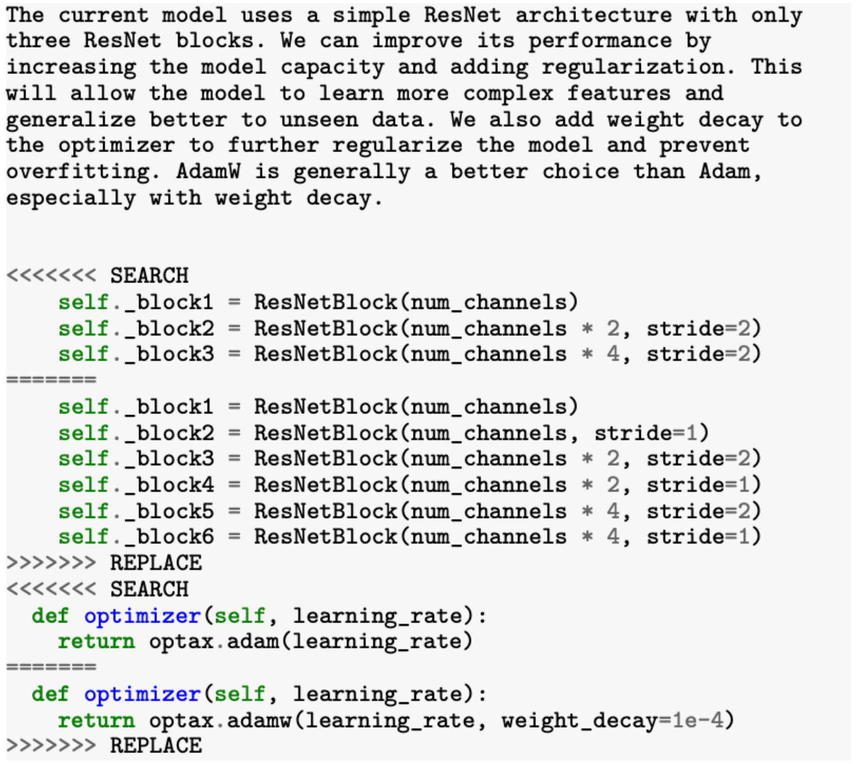
```
Act as an expert software developer. Your task is to iteratively
improve the provided codebase. [...]

- 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.
```

(b)

Step 3: LLM to generate changes to code



```
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
```

(c)

- Optional: Can also evolve entire code block if code is short

Step 4: Evaluation

- User can provide test cases to test out the solution obtained. Test cases can be in increasing difficulty and improvement be made on next level once earlier levels are solved (curriculum learning)
- User-defined criterion that LLM can evaluate on
- Parallel evaluation on multiple initialisations / input sets to speed up inference

Step 5: Evolution

- AlphaEvolve continually generates a growing number of solutions with evaluation results (scores and program outputs) attached to them

- These solutions are stored in an **evolutionary database**, the primary goal of which is to optimally resurface previously explored ideas in future generations.

Additional Details

Different levels of abstraction space to evolve

- Can evolve in different abstraction spaces, e.g.:
  - string representation (as in classical evolutionary algorithms)
  - evolve a function of a definite form that specifies how to construct the solution from scratch
  - evolve a bespoke search algorithm to find the solution within some fixed compute budget
  - Co-evolve search algorithms and intermediate solutions together

Prompt Sampling

- This prompt comprises multiple previously discovered solutions sampled from the program database, as well as system instructions on how to propose changes to a particular solution

- Additionally, there are:
  - **Explicit context:** details about the problem being solved, such as fixed human-written instructions, equations, code snippets, or relevant literature (e.g., pdf files)
  - **Stochastic formatting:** template placeholders with human-provided alternatives for increased diversity, instantiated using probability distributions provided in a separate config file
  - **Rendered evaluation results:** usually this will include a program, the result of executing that program, and the scores assigned by the evaluate function
  - **Meta prompt evolution:** instructions and context suggested by the LLM itself in an additional prompt-generation step, co-evolved in a separate database analogous to the solution programs

Ablation shows all components are important

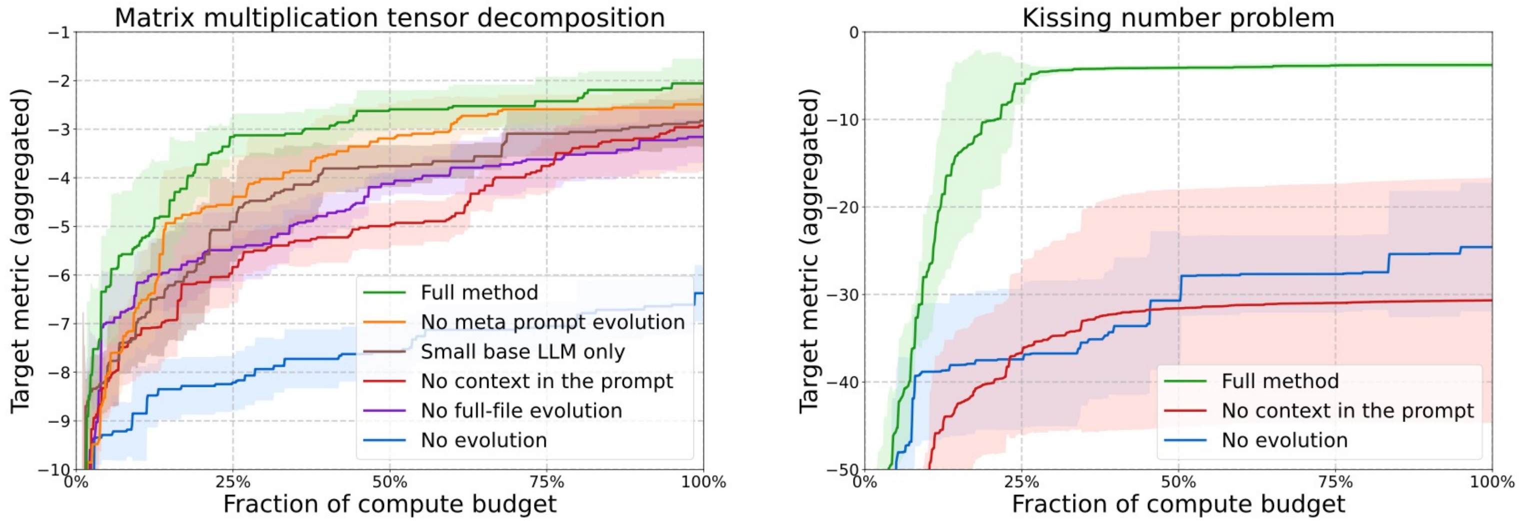


Figure 8 | Left: Ablations of *AlphaEvolve* on the problem of finding low-rank tensor decomposition for faster matrix multiplication. Right: Ablations of *AlphaEvolve* on the problem of finding sphere packings for improving kissing numbers. Each curve shows the performance of an individual setting with increasing compute budget, averaged over all considered targets (higher values on the target metric are better). The shades indicate intra-target standard deviation, averaged over three independent runs of *AlphaEvolve*, initialized with different random seeds.

Question to Ponder

- Can AlphaEvolve be used to evolve academic papers (e.g. AI Scientist), or for the ARC-AGI challenge?
- What if there is no objective evaluation function for the problem, e.g. essay writing, music creation?
- Is code enough to model the possible action spaces of problems? How can this be extended to more action spaces?