FunSearch & AlphaEvolve

Bartosz Piotrowski

August 21, 2025
The Reasoning Reading Group @ FAIR

Motivation

Preconditions:

- A problem of finding optimal heuristic / program / function.
- A pre-trained, coding-capable LLM.
- An automated evaluator returning a scalar score.

Motivation

Preconditions:

- A problem of finding optimal heuristic / program / function.
- A pre-trained, coding-capable LLM.
- An automated evaluator returning a scalar score.

The simplest strategy:

- generating multiple independent samples from the LLM with non-zero temperature,
- evaluating all of them,
- selecting the best one.

Motivation

Preconditions:

- A problem of finding optimal heuristic / program / function.
- A pre-trained, coding-capable LLM.
- An automated evaluator returning a scalar score.

The simplest strategy:

- generating multiple independent samples from the LLM with non-zero temperature,
- evaluating all of them,
- selecting the best one.

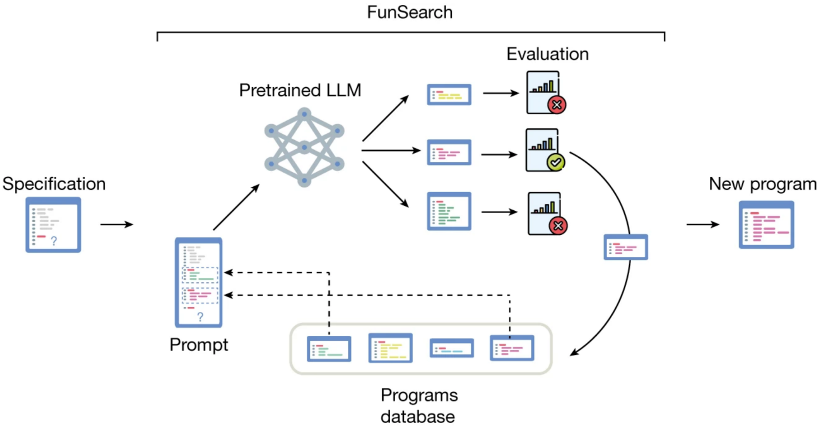
But instead of sampling independently, can we **incorporate the evaluator feedback** into subsequent generations?

FunSearch

Key ingredients:

- *best-shot* prompting,
- a growing database of programs,
- an evolutionary strategy acting on it.

FunSearch


FunSearch
Evaluation
Pretrained LLM
Specification
New program
Prompt
Programs
database

Database of programs

- Several islands/subpopulations growing independently.
- Higher-scoring programs, but also shorter ones, are prioritized.
- Less good programs are eventually discarded.
- Different islands are mixed with each other to an extent.
- Why multiple islands? For diversity.

Best-shot prompting

- k good programs per prompt sampled, for each island.
- Information which one is better incorporated into the prompt (*v0*, *v1*, ...).
- In the actual experiments, $k = 2$.

Other details

- **Model:**
  - Codey, a PaLM2 model fine-tuned on code.
  - Smaller, faster-inference model chosen.
- Implementation with **three asynchronous workers:**
  - database,
  - generator,
  - evaluator.
- Prompting with **templates / skeletons of programs**.
  - The LLM asked to modify only the essential function.

Problems tackled

Two known combinatorics problems:

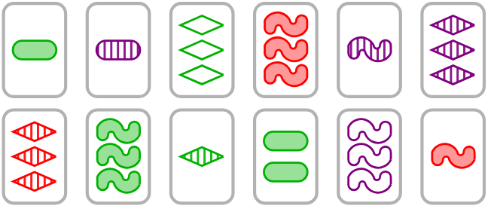
- Cap set problem.
- (Online) bin packing.

Cap set problem

What is the largest possible set of vectors in $\mathbb{Z}_3^n$ (*cap set*) such that no three vectors sum to zero?

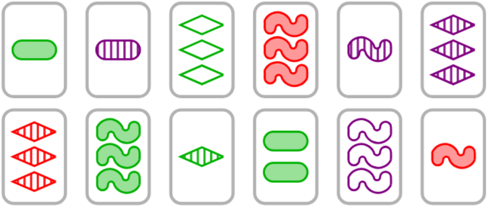
Cap set problem

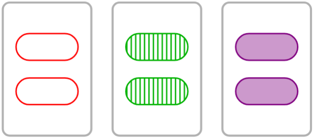
What is the largest possible set of vectors in $\mathbb{Z}_3^n$ (*cap set*) such that no three vectors sum to zero?



Cap set problem

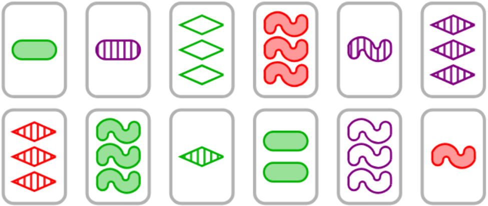
What is the largest possible set of vectors in $\mathbb{Z}_3^n$ (*cap set*) such that no three vectors sum to zero?

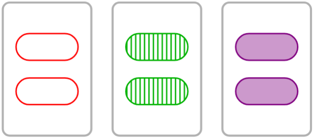




Cap set problem

What is the largest possible set of vectors in $\mathbb{Z}_3^n$ (*cap set*) such that no three vectors sum to zero?





n	3	4	5	6	7	8
Best known	9	20	45	112	236	496
FunSearch	9	20	45	112	236	512

Cap set problem template

```
"""Finds large cap sets."""
import numpy as np
import utils_capset

# Function to be executed by FunSearch.
def main(n):
  """Runs `solve` on `n`-dimensional cap set and
  ↪  evaluates the output."""
  solution = solve(n)
  return evaluate(solution, n)

def evaluate(candidate_set, n):
  """Returns size of candidate_set if it is a cap
  ↪  set, None otherwise."""
  if utils_capset.is_capset(candidate_set, n):
    return len(candidate_set)
  else:
    return None

def solve(n):
  """Builds a cap set of dimension `n` using
  ↪  `priority` function."""
  # Precompute all priority scores.
  elements = utils_capset.get_all_elements(n)
  scores = [priority(el, n) for el in elements]
  # Sort elements according to the scores.
  elements = elements[np.argsort(scores,
  ↪  kind='stable')[::-1]]

  # Build `capset` greedily, using scores for
  ↪  prioritization.
  capset = []
  for element in elements:
    if utils_capset.can_be_added(element, capset):
      capset.append(element)
  return capset

# Function to be evolved by FunSearch.
def priority(element, n):
  """Returns the priority with which we want to add
  ↪  `element` to the cap set."""
  return 0.0
```

Cap set solution – the priority function

```
def priority(el: tuple[int,...],
↪  n: int) -> float:
  score = n
  in_el = 0
  el_count = el.count(0)

  if el_count == 0:
    score += n**2
    if el[1] == el[-1]:
      score *= 1.5
    if el[2] == el[-2]:
      score *= 1.5
    if el[3] == el[-3]:
      score *= 1.5
  else:
    if el[1] == el[-1]:
      score *= 0.5
    if el[2] == el[-2]:
      score *= 0.5

  for e in el:
    if e == 0:
      if in_el == 0:
        score *= n * 0.5
      elif in_el == el_count - 1:
        score *= 0.5
      else:
        score *= n * 0.5 ** in_el
      in_el += 1
    else:
      score += 1

  if el[1] == el[-1]:
    score *= 1.5
  if el[2] == el[-2]:
    score *= 1.5

  return score
```

AlphaEvolve

tldr: FunSearch scaled-up in multiple dimensions.

FunSearch [83]	*AlphaEvolve*
evolves single function	evolves entire code file
evolves up to 10-20 lines of code	evolves up to hundreds of lines of code
evolves code in Python	evolves any language
needs fast evaluation ($\leq$ 20min on 1 CPU)	can evaluate for hours, in parallel, on accelerators
millions of LLM samples used	thousands of LLM samples suffice
small LLMs used; no benefit from larger	benefits from SOTA LLMs
minimal context (only previous solutions)	rich context and feedback in prompts
optimizes single metric	can simultaneously optimize multiple metrics

AlphaEvolve – notable differences

- Larger base LLMs (Gemini 2.0 Flash, Gemini 2.0 Pro).
- Much richer prompts:
  - context for the problem provided,
  - detailed evaluation results included,
  - meta prompt evolution.
- Output format: a **diff** rather than a new program version.
- Evaluation:
  - many evaluators,
  - including LLM judges.
- Improved evolution strategy:
  - different islands (as before),
  - MAP elites algorithm.

Problems tackled

50 mathematical problems:

- Fast matrix multiplication, kissing number problem, ...
- For 75% problems AlphaEvolve matched the optimal known solution.
- For 20% problems AlphaEvolve found better solution.

Also practical problems related to computational infrastructure:

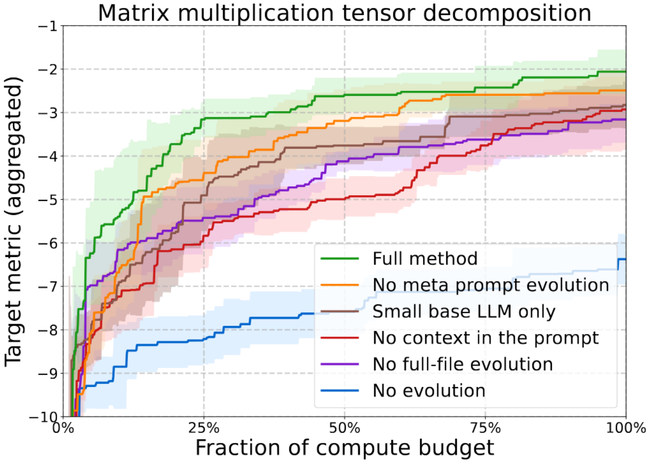
- Scheduling jobs on a cluster, optimizing JAX kernel, ...
- Some of the solutions found by AlphaEvolve went into production at Google.

Optimizing matrix multiplications

- The simple algorithm for multiplying $n \times m$ and $m \times k$ matrices requires nmk scalar multiplications.
- But Volker Strassen in 1969 showed it can be done with fewer multiplications.
- The optimal algorithm in general not known.
- AlphaEvolve found improvements for multiple specific cases.

$\langle m, n, p \rangle$	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

Ablations



Matrix multiplication tensor decomposition
Target metric (aggregated)
−1
−2
−3
−4
−5
−6
−7
−8
−9
−10
0%
25%
50%
75%
100%
Fraction of compute budget
Full method
No meta prompt evolution
Small base LLM only
No context in the prompt
No full-file evolution
No evolution

Closing remarks

- FunSearch/AlphaEvolve can be seen as:
  - mechanizing effective interaction with the LLM,
  - complex meta-generation framework.[1]
- An interesting trade-off:
  - smaller, less clever, but faster LLM = more samples (FunSearch) *vs*
  - larger, more clever, but slower LLM = higher-quality samples (AlphaEvolve).
- Is it possible to adapt the AlphaEvolve-style framework to formal proving?
  - A notion of *partial proof progress* would be needed.
  - But isn't it similar to what Seed Proved does at test time?
- AlphaEvolve could be distilled into a better base LLM, which would result in *meta-evolution*.
- Open-source implementation: OpenEvolve.

[1]See a nice NeurIPS tutorial: *Beyond Decoding: Meta-Generation Algorithms for Large Language Models*