Evolutionary Algorithms and LLMs

28 Oct. 2025, Reasoning in AI

Topics of the second half

- Evolutionary algorithms (“survival of the fittest” type of algorithm)
- Applications of LLMs in this framework
- A little bit of RL and tool calling

Example Applications

Article

Mathematical discoveries from program search with large language models

https://doi.org/10.1038/s41586-023-06924-6
Received: 12 August 2023
Accepted: 30 November 2023
Published online: 14 December 2023
Open access
Check for updates

Bernardino Romera-Paredes[1,4]✉, Mohammadamin Barekatain[1,4], Alexander Novikov[1,4], Matej Balog[1,4], M. Pawan Kumar[1,4], Emilien Dupont[1,4], Francisco J. R. Ruiz[1,4], Jordan S. Ellenberg[2], Pengming Wang[1], Omar Fawzi[3], Pushmeet Kohli[1]✉ & Alhussein Fawzi[1,4]✉

Large language models (LLMs) have demonstrated tremendous capabilities in solving complex tasks, from quantitative reasoning to understanding natural language. However, LLMs sometimes suffer from confabulations (or hallucinations), which can

Algorithm Discovery With LLMs: Evolutionary Search Meets Reinforcement Learning

Anja Surina[1]*, Amin Mansouri[1], Lars Quaedvlieg[1], Amal Seddas[1], Maryna Viazovska[1], Emmanuel Abbe[1,2], Caglar Gulcehre[1]
[1]EPFL [2]Apple

Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model

Fei Liu [1] Xialiang Tong [2] Mingxuan Yuan [2] Xi Lin [1] Fu Luo [3] Zhenkun Wang [3] Zhichao Lu [1] Qingfu Zhang [1]

Evolutionary Algorithms

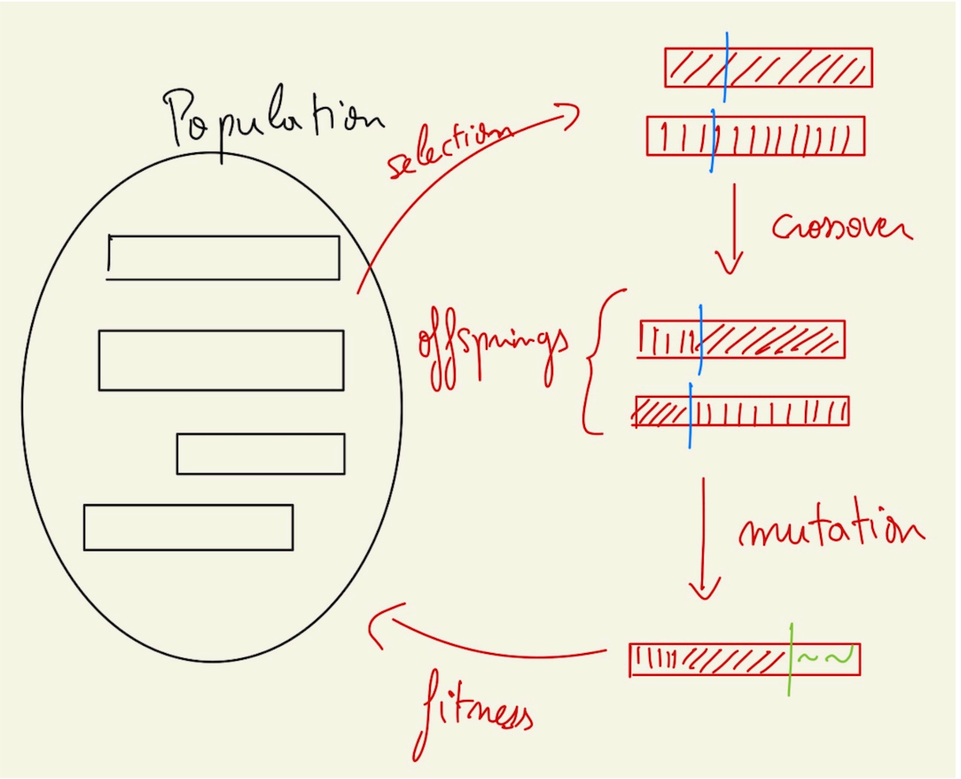
Evolutionary Algorithms

- Basic ideas: Adaptation is intelligence, apply the concept of “survival of the fittest” for algorithms
- How can we use this idea for optimization?
- More precisely, say we want to optimize some complicated function, like

$$f(x_1, x_2) = x_1^2 + x_1 x_2 \cdot \sin(x_2)$$

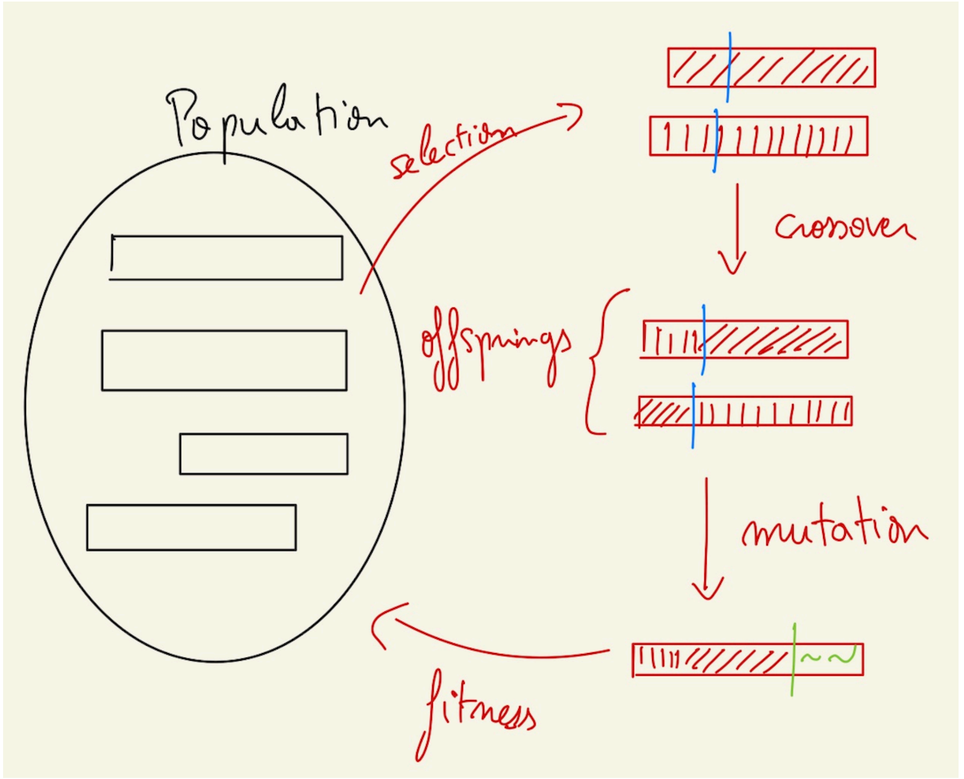
Simple Genetic Algorithm

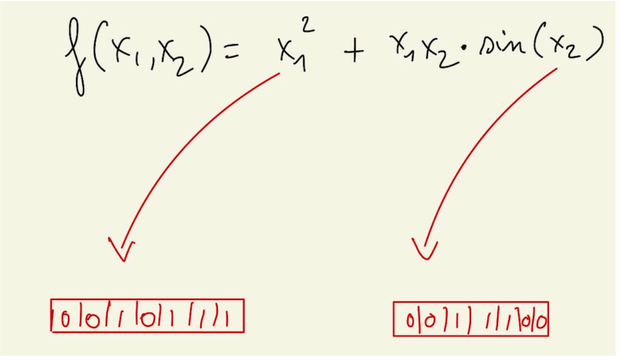
1. Initialize population
2. Calculate the **fitness** of your population
3. While stopping criterion not satisfied:
    1. Select parents
    2. Perform crossover -> offsprings
    3. Apply mutation
    4. Calculate total fitness
    5. If offsprings are fit, add them to population



Simple Genetic Algorithm

1. Initialize population
2. Calculate the **fitness** of your population
3. While stopping criterion not satisfied:
    1. Select parents
    2. Perform crossover -> offsprings
    3. Apply mutation
    4. Calculate total fitness
    5. If offsprings are fit, add them to population





Pros and cons of Genetic Algorithms

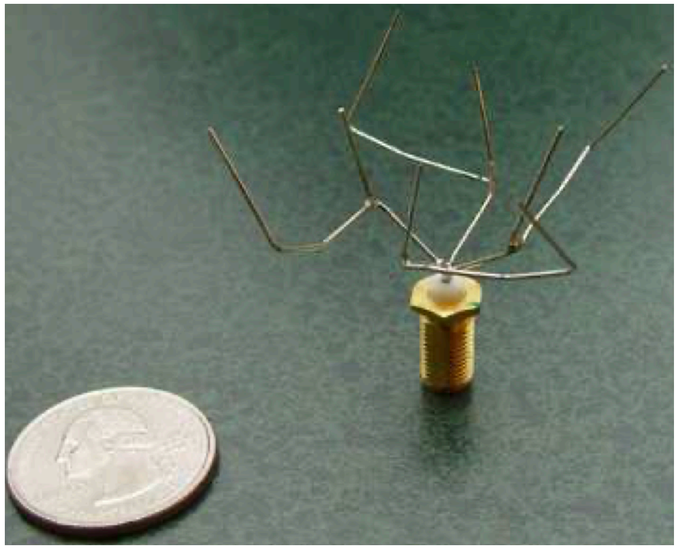
Pros:

- Simple to implement
- Provide many solutions (good if there are local extrema)
- Can be parallelized

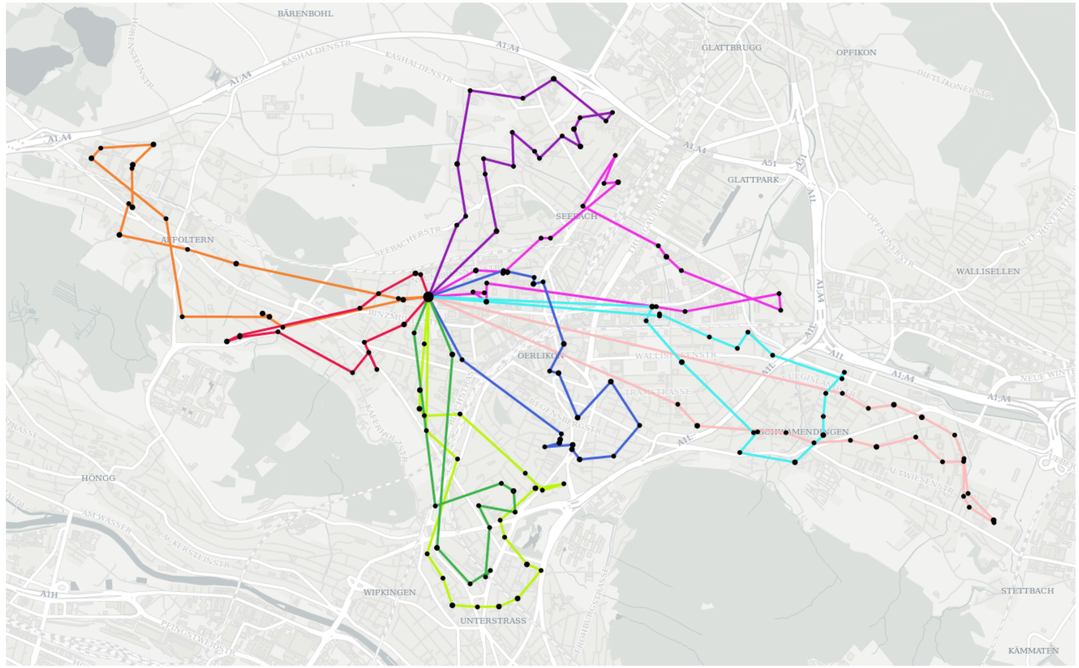
Cons:

- Can be slow (evolution is slow)
- Fitness function can be difficult to design

Applications of Genetic Algorithms



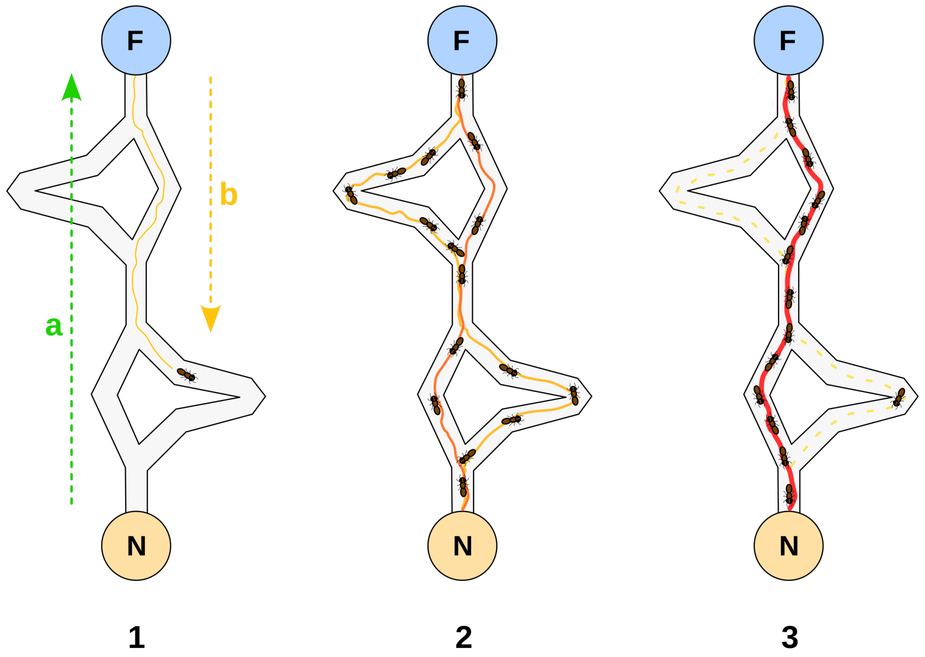
Engineering



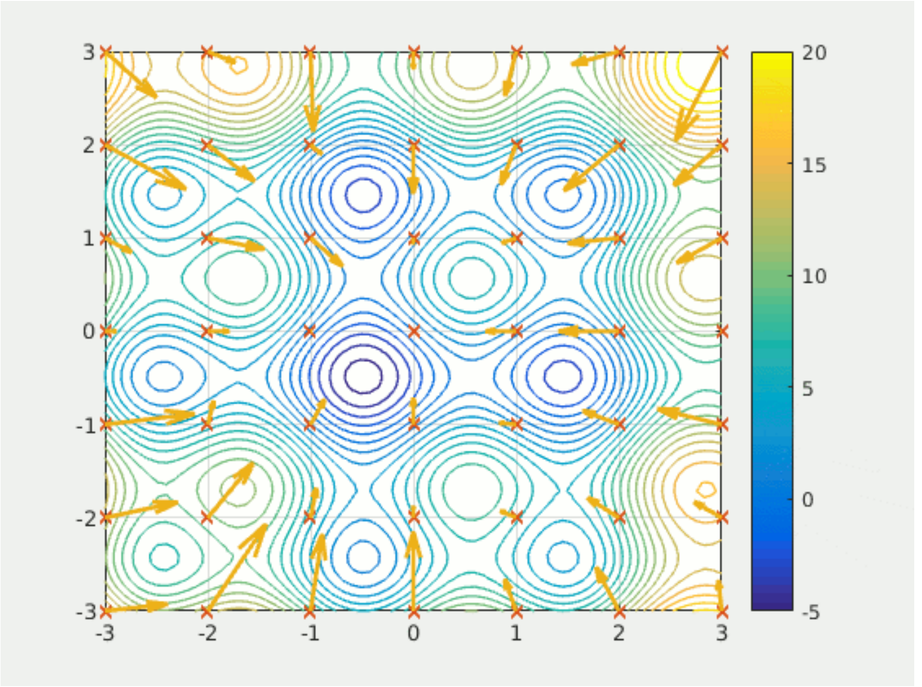
Combinatorial Optimization

More applications in Finance, Biology, etc

More nature inspired algorithms



Ant colony optimization



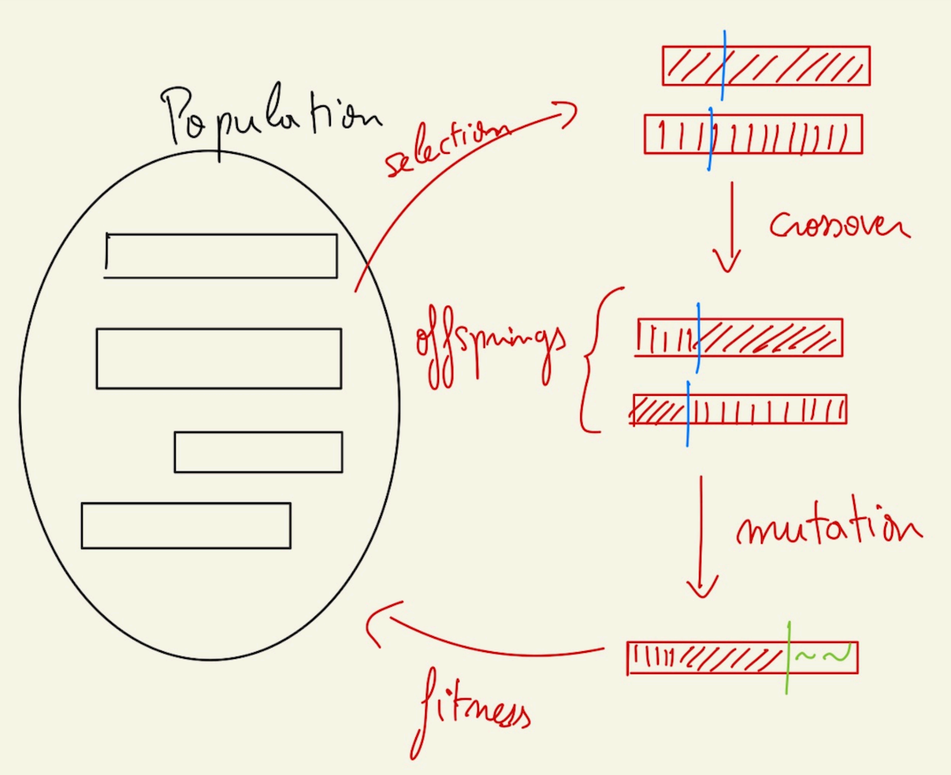
Swarm intelligence

+ Neural networks, and more

LLMs and Evolution

LLM and Evolution

Idea: use LLMs as the mutation/reproduction operator

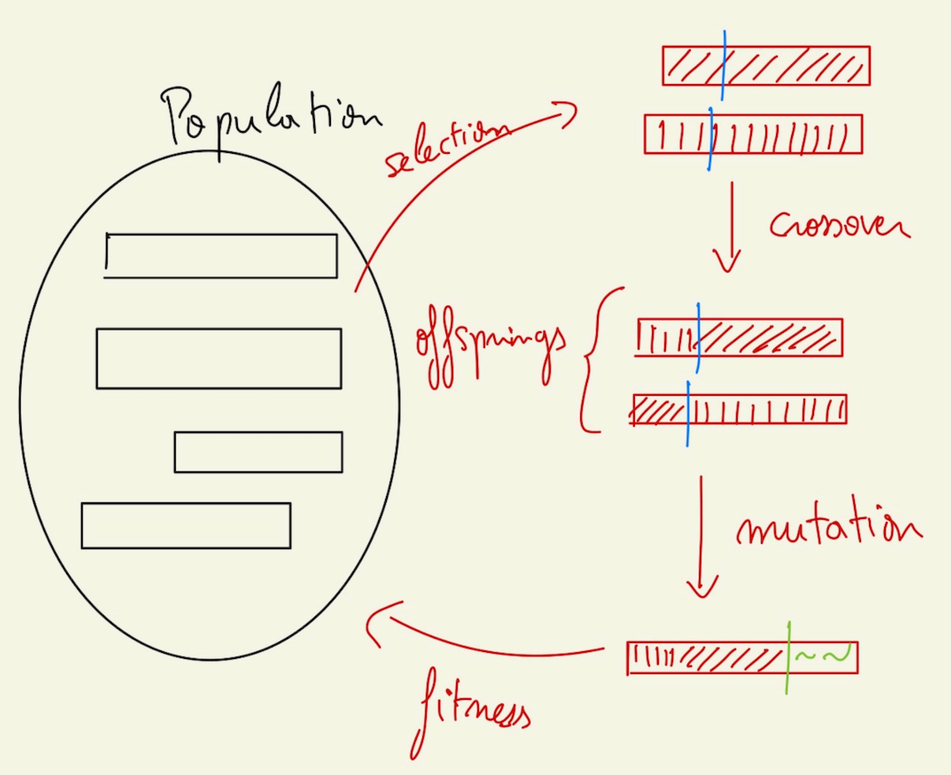


One big advantage of this:
LLMs speak our language!

Rest of the lecture = applying this method to math problems

LLM and Evolution

Idea: use LLMs as the mutation/reproduction operator

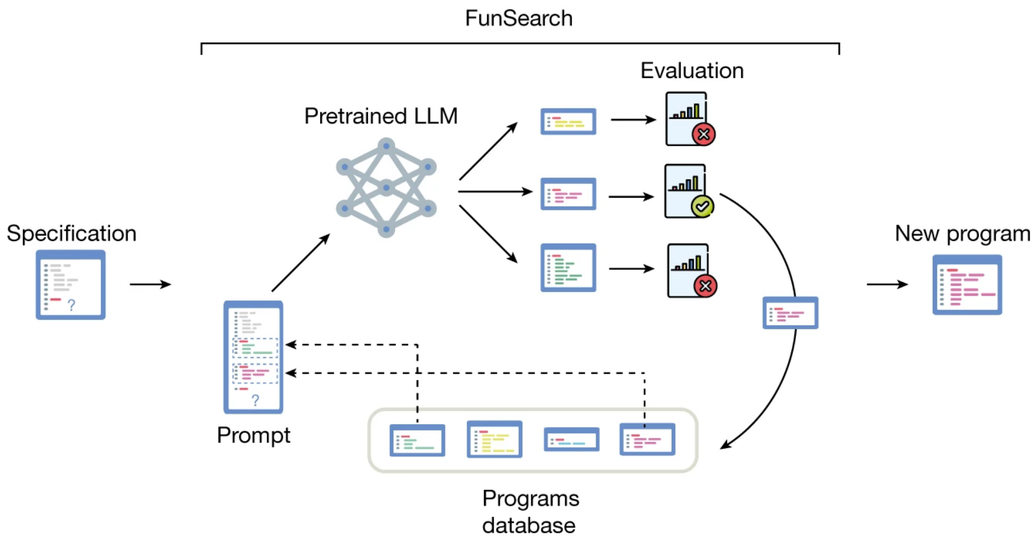


One key point: we will ask the LLM to solve a problem hard but easy to **verify**

Ask the LLM to generate Python code that we can run to verify - tool calling

LLM and Evolution

Idea: use LLMs as the mutation/reproduction operator



One key point: we will ask the LLM to solve a problem hard but easy to **verify**

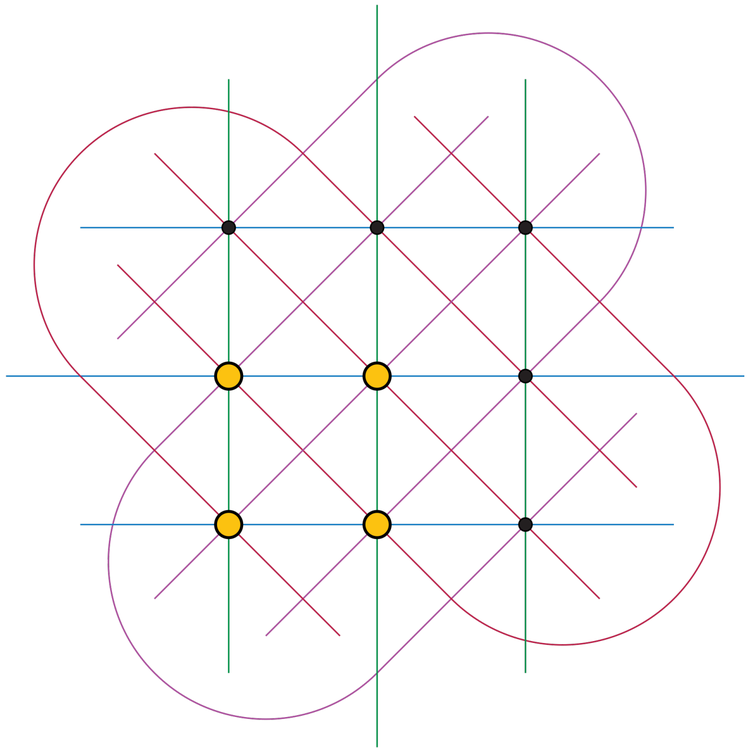
Ask the LLM to generate Python code that we can run to verify - tool calling

Safeguard against hallucinations

Case study: Problem 1

The “cap set problem”: Find the biggest set S of elements of $\mathbb{Z}_3^n$ such that no three elements of S sum up to 0 (i.e. $x_1 + x_2 + x_3 \neq 0$ for all distinct $x_1, x_2, x_3 \in S$)

Example for $n = 2$



Case study: Problem 1

The “cap set problem”: Find the biggest set S of elements of $\mathbb{Z}_3^n$ such that no three elements of S sum up to 0 (i.e. $x_1 + x_2 + x_3 \neq 0$ for all distinct $x_1, x_2, x_3 \in S$)

How big can S be as a function of n?

We know that $2.2202^n \leq |S| \leq 2.756^n$

Lower bound found thanks to LLMs

Case study: Problem 1

Strategy for building a big S:

1. Define a priority function $\mathbb{Z}_3^n \mapsto \mathbb{R}$
2. Add element one by one to S, in the order given by priority function.

Case study: Problem 1

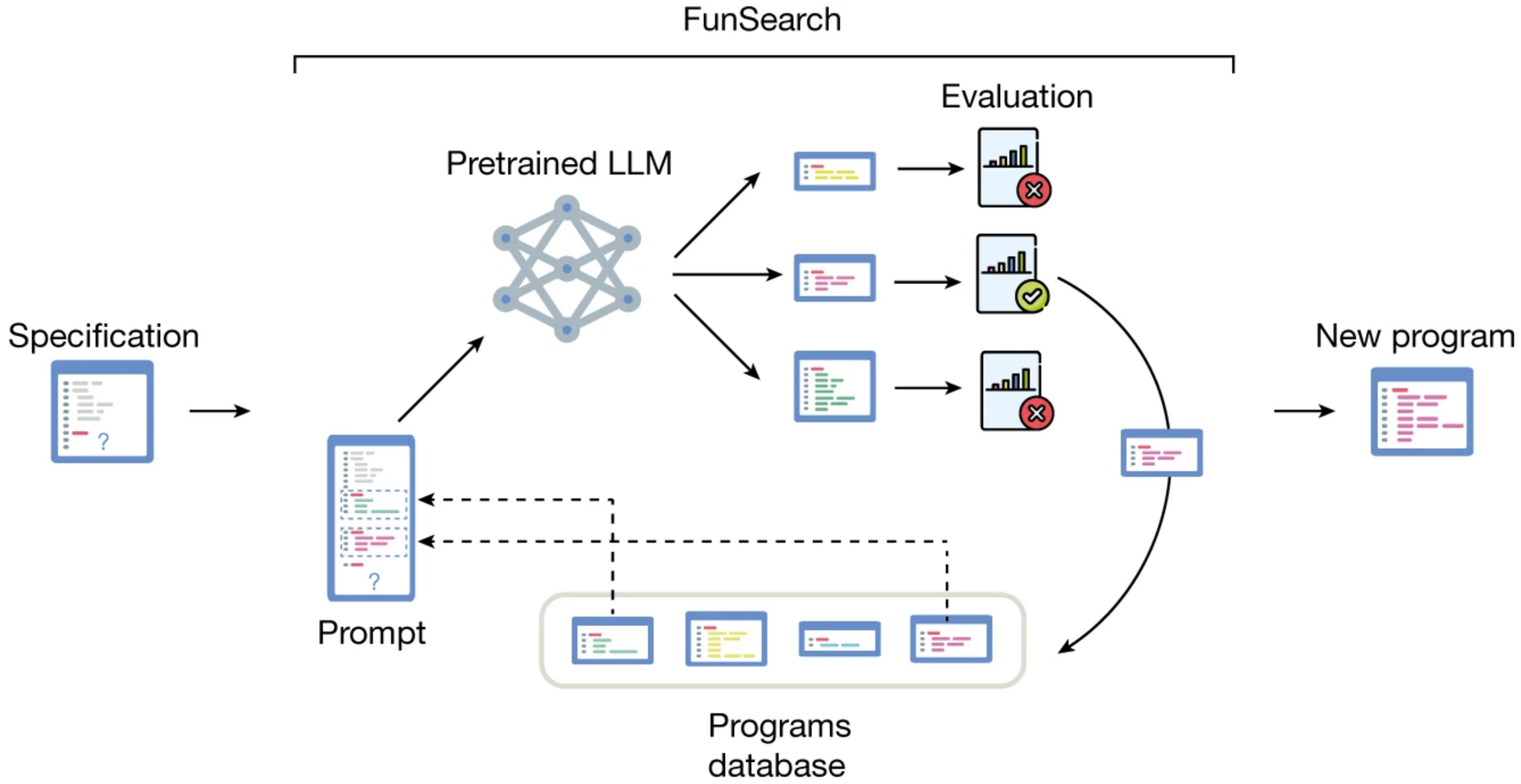
Strategy for building a big S:

1. Define a priority function $\mathbb{Z}_3^n \mapsto \mathbb{R}$
2. Add element one by one to S, in the order given by priority function.

How can we find a good priority function?

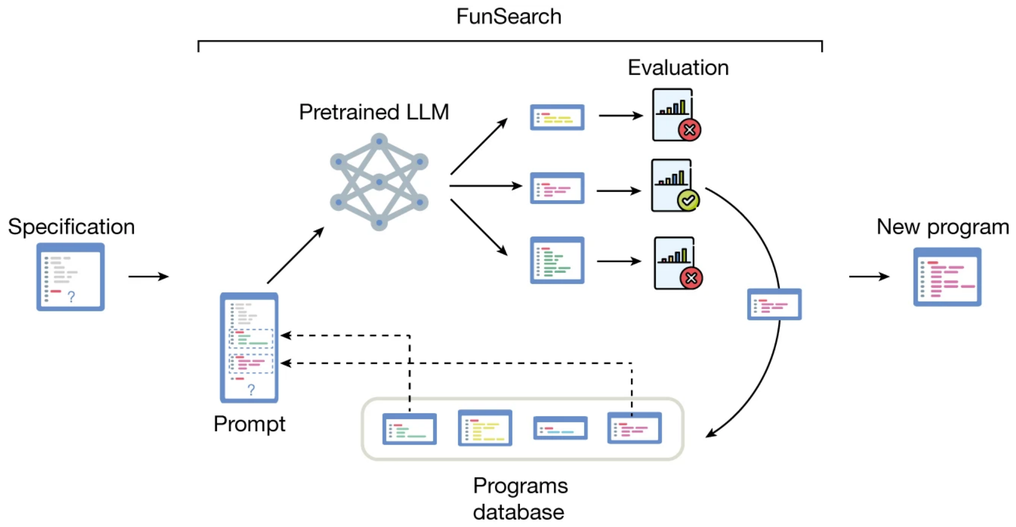
Case study: Problem 1

How can we find a good priority function?



Case study: Problem 1

How can we find a good priority function?



Prompt:

```
"""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_v0(el, n) for el in elements]
    # Sort elements according to the scores (descending, stable).
    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_v0(element, n):
    """Returns the priority with which we want to add `element` to the cap set."""
    return 0.0

def priority_v1(element, n):
    """Improved version of priority_v0."""
```

Case study: Problem 1

Results:

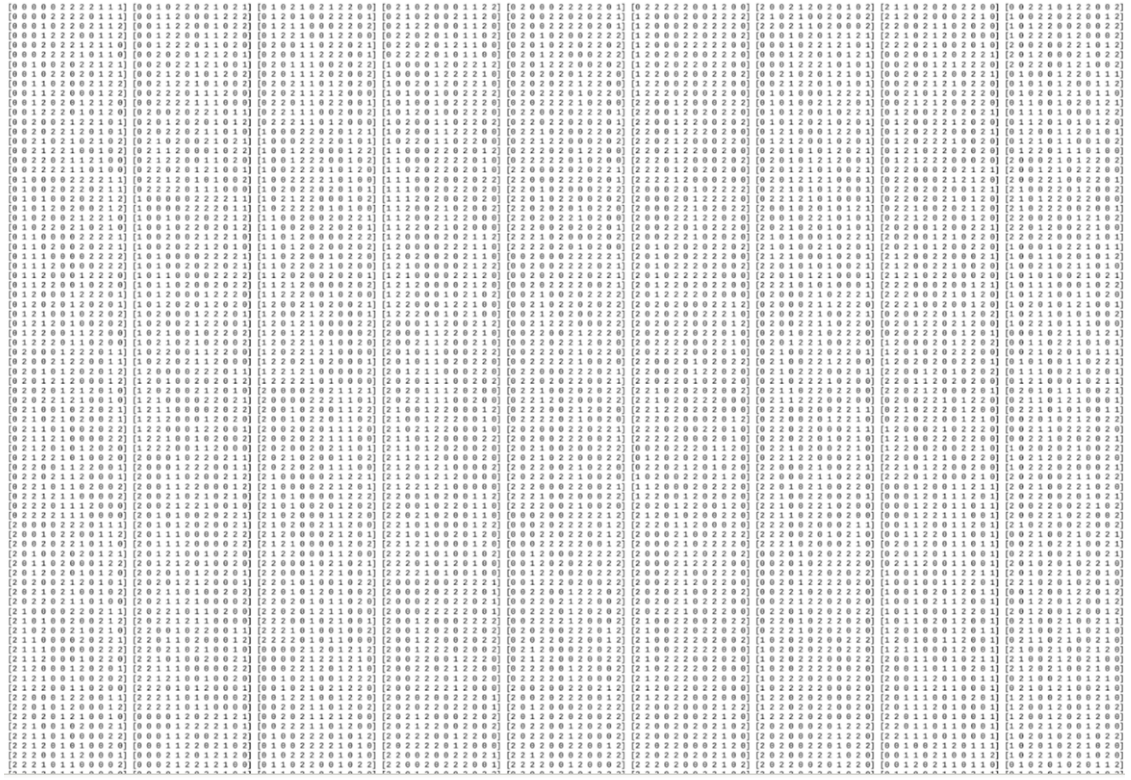
n	3	4	5	6	7	8
Best known	9	20	45	112	236	496
FunSearch	9	20	45	112	236	512

From Romera-Paredes et al., Nature, 2024

```
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
```

Case study: Problem 1

Results:



From Romera-Paredes et al., Nature, 2024

```python
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
```

Case study: Problem 1

One advantage over classical algorithm:

- Much more “explainable”. The authors of the paper managed to obtain new insights on the problem (still with the help of an **expert** on the problem).

Downsides:

- Can be expensive to run (here around $2 \cdot 10^6$ programs were generated, 15 GPUs + 150 CPUs for approx 2 days). LLM was PaLM 2 (probably 340B parameters in 2023).
- Not always clear that it is applicable (also true for classical algorithms)
- Sometimes difficult to force the LLM to “be creative”

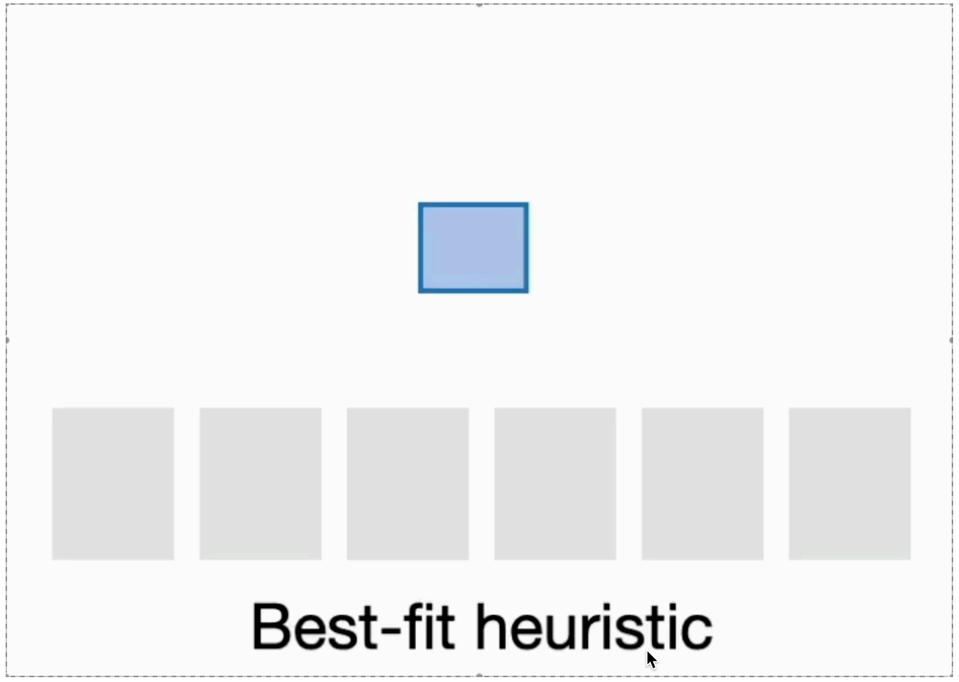
Case study: Problem 2

Online bin packing: Items arrive one by one with size s_i for item i. We have bins of capacity 1, and we want to pack each item into a bin without exceeding the capacity of the bin.

How to do this in order to minimize the number of bins needed?

Case study: Problem 2

Online bin packing: Items arrive one by one with size s_i for item i. We have bins of capacity 1, and we want to pack each item into a bin without exceeding the capacity of the bin.



Case study: Problem 2

Algorithm strategy:

1. Define a priority function over bins (given the current state).
2. When element i arrives, place it in the highest priority bin where it fits.

Case study: Problem 2

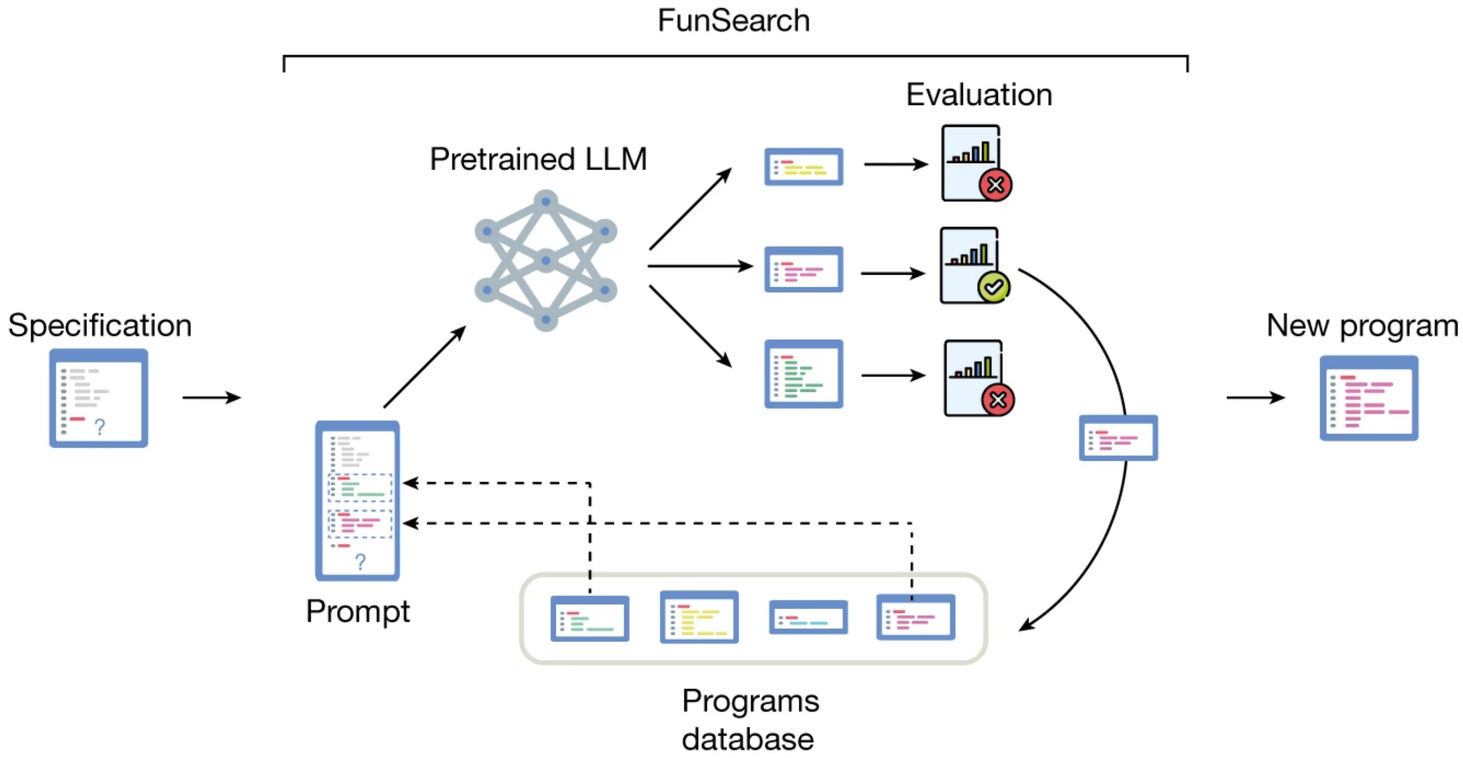
Algorithm strategy:

1. Define a priority function over bins (given the current state).
2. When element i arrives, place it in the highest priority bin where it fits.

How can we find a good priority function?

Case study: Problem 2

How can we find a good priority function?



Other idea from Surina et al. COLM 2025: add RL in the loop

Case study: Problem 2

Results for FunSearch (without RL):

	OR1	OR2	OR3	OR4	Weibull 5k	Weibull 10k	Weibull 100k
First Fit	6.42%	6.45%	5.74%	5.23%	4.23%	4.20%	4.00%
Best Fit	5.81%	6.06%	5.37%	4.94%	3.98%	3.90%	3.79%
FunSearch	**5.30%**	**4.19%**	**3.11%**	**2.47%**	**0.68%**	**0.32%**	**0.03%**

From Romera-Paredes et al., Nature, 2024

```python
def heuristic(item: float, bins: np.ndarray) -> np.ndarray:
  """Online bin packing heuristic discovered with FunSearch."""
  score = 1000 * np.ones(bins.shape)
  # Penalize bins with large capacities.
  score -= bins * (bins - item)
  # Extract index of bin with best fit.
  index = np.argmin(bins)
  # Scale score of best fit bin by item size.
  score[index] *= item
  # Penalize best fit bin if fit is not tight.
  score[index] -= (bins[index] - item)**4
  return score
```

Case study: Problem 2

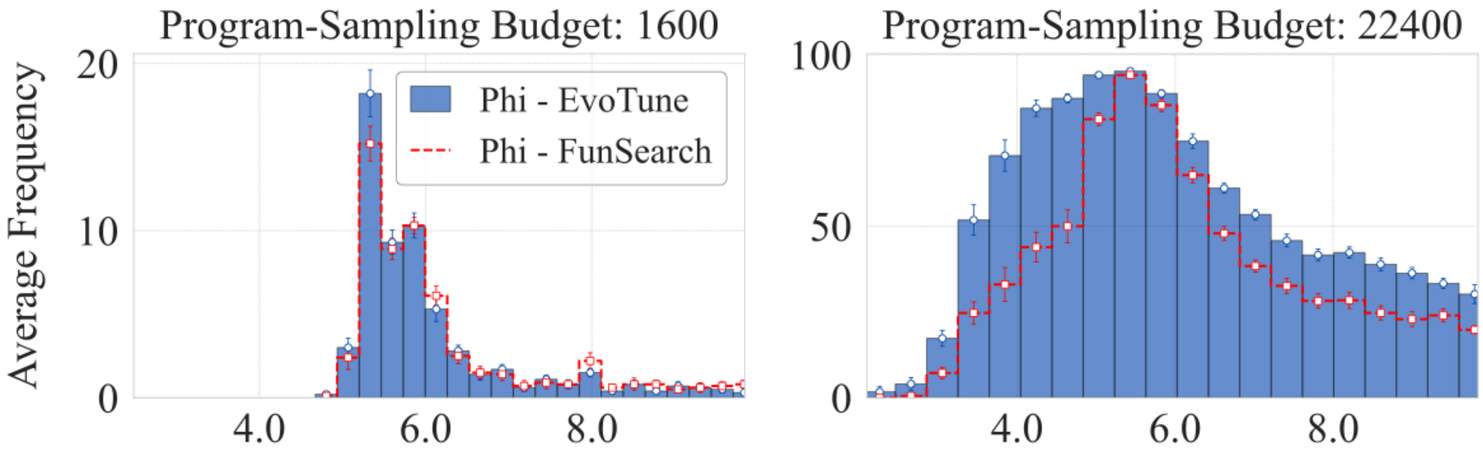
EvoTune (with RL) vs FunSearch (without RL):

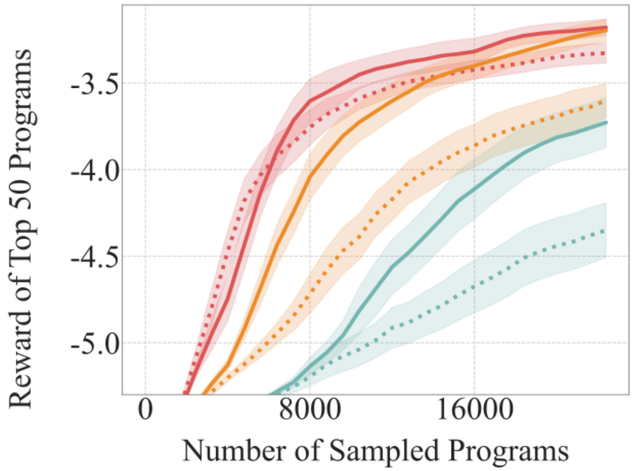
	Validation Set			Validation-Perturbed Set			Test Set		
	9.6K	16K	22.4K	9.6K	16K	22.4K	9.6K	16K	22.4K
Bin Packing									
Llama FunSearch	5.08 ± 0.09	4.67 ± 0.15	4.35 ± 0.16	4.46 ± 0.14	4.01 ± 0.19	3.68 ± 0.18	4.52 ± 0.15	4.07 ± 0.19	3.77 ± 0.18
Llama EvoTune	**4.96 ± 0.09**	**4.11 ± 0.17**	**3.73 ± 0.15**	**4.31 ± 0.15**	**3.39 ± 0.20**	**3.10 ± 0.17**	**4.39 ± 0.15**	**3.51 ± 0.19**	**3.21 ± 0.14**
Phi FunSearch	4.47 ± 0.13	3.86 ± 0.12	3.60 ± 0.10	3.89 ± 0.18	3.34 ± 0.14	3.09 ± 0.11	3.99 ± 0.17	3.42 ± 0.13	3.19 ± 0.09
Phi EvoTune	**3.81 ± 0.12**	**3.40 ± 0.08**	**3.12 ± 0.07**	**3.31 ± 0.17**	**2.88 ± 0.11**	**2.70 ± 0.01**	**3.38 ± 0.17**	**2.99 ± 0.12**	**2.80 ± 0.10**
Granite FunSearch	3.64 ± 0.08	3.42 ± 0.05	3.33 ± 0.06	3.12 ± 0.09	2.95 ± 0.07	2.86 ± 0.08	3.20 ± 0.07	3.05 ± 0.06	2.97 ± 0.08
Granite EvoTune	**3.50 ± 0.10**	**3.32 ± 0.08**	**3.18 ± 0.05**	**2.92 ± 0.14**	**2.75 ± 0.08**	**2.66 ± 0.07**	**3.07 ± 0.12**	**2.89 ± 0.07**	**2.82 ± 0.07**

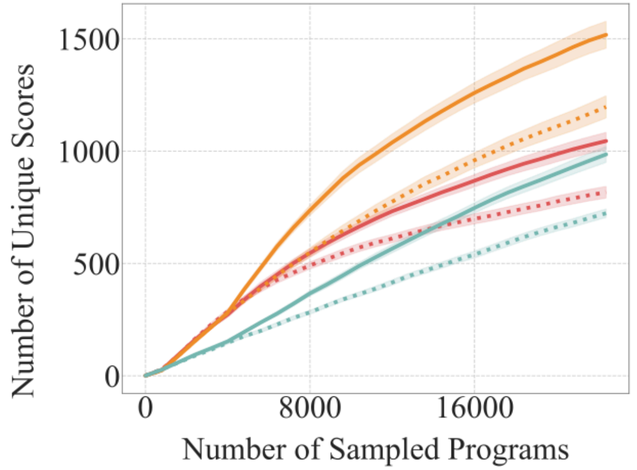
From Surina et al., COLM, 2025

Case study: Problem 2

EvoTune (with RL) vs FunSearch (without RL):







Use of RL promotes diversity

From Surina et al., COLM, 2025

Little Detour into RL

Use of RL promotes diversity

Different flavors of RL

$$\max_{\theta} \; \mathbb{E}_{y\sim\pi_\theta}\big[R(y)\big] \; - \; \beta\, D_{\mathrm{KL}}\big(\pi_\theta \,\|\, \pi_{\mathrm{ref}}\big)\,. \qquad \text{Reverse KL}$$

$$\max_{\theta} \; \mathbb{E}_{y\sim\pi_\theta}\big[R(y)\big] \; - \; \beta\, D_{\mathrm{KL}}\big(\pi_{\mathrm{ref}} \,\|\, \pi_\theta\big)\,. \qquad \text{Forward KL}$$

where

$$D_{\mathrm{KL}}(P\|Q) = \sum_{x\in\mathcal{X}} P(x)\,\log\frac{P(x)}{Q(x)}, \quad \text{with } Q(x) > 0 \text{ whenever } P(x) > 0.$$

Little Detour into RL

Use of RL promotes diversity

Different flavors of RL

$$\max_{\theta} \; \mathbb{E}_{y\sim\pi_\theta}\big[R(y)\big] \; - \; \beta\, D_{\mathrm{KL}}\big(\pi_\theta \,\|\, \pi_{\mathrm{ref}}\big)\,. \qquad \text{Reverse KL}$$

$$\max_{\theta} \; \mathbb{E}_{y\sim\pi_\theta}\big[R(y)\big] \; - \; \beta\, D_{\mathrm{KL}}\big(\pi_{\mathrm{ref}} \,\|\, \pi_\theta\big)\,. \qquad \text{Forward KL}$$

Reverse KL: $-\beta H(\pi_\theta) + \beta \mathbb{E}_{\pi_\theta}(\log(\pi_{\mathsf{ref}}))$

Forward KL: $\beta \mathbb{E}_{\pi_{\mathsf{ref}}}(\log(\pi_\theta))$

Little Detour into RL

Reverse KL: $-\beta H(\pi_\theta) + \beta \mathbb{E}_{\pi_\theta}(\log(\pi_{\text{ref}}))$

Forward KL: $\beta \mathbb{E}_{\pi_{\text{ref}}}(\log(\pi_\theta))$

Reverse KL: If we have $\pi_{\text{ref}}(x) = 0$ and $\pi_\theta(x) > 0$ then we pay a huge penalty

Mode seeking

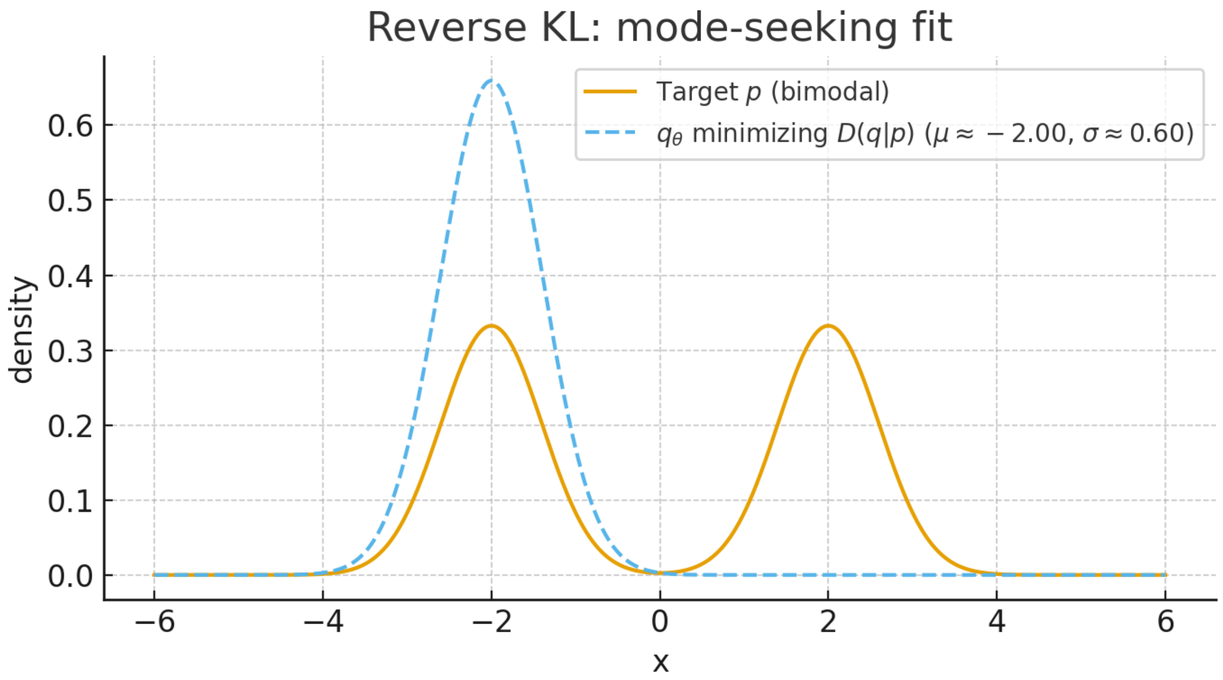
Forward KL: If we have $\pi_{\text{ref}}(x) > 0$ and $\pi_\theta(x) = 0$ then we pay a huge penalty

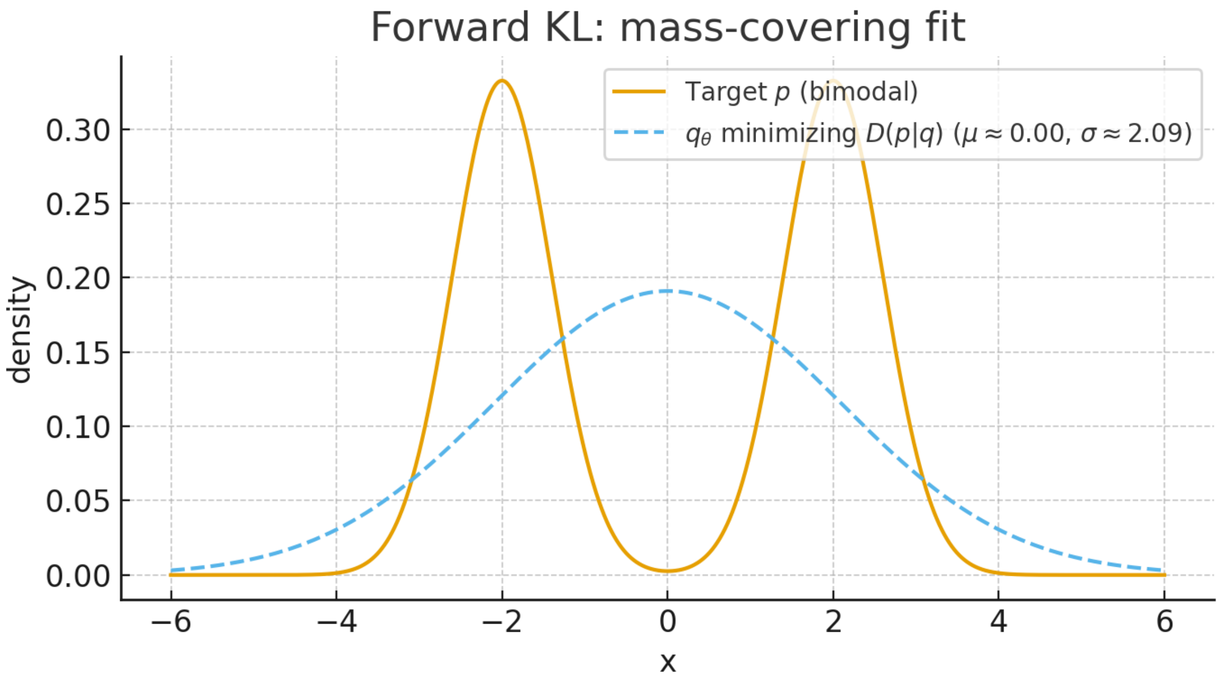
Mass covering

Little Detour into RL

Reverse KL: Mode seeking

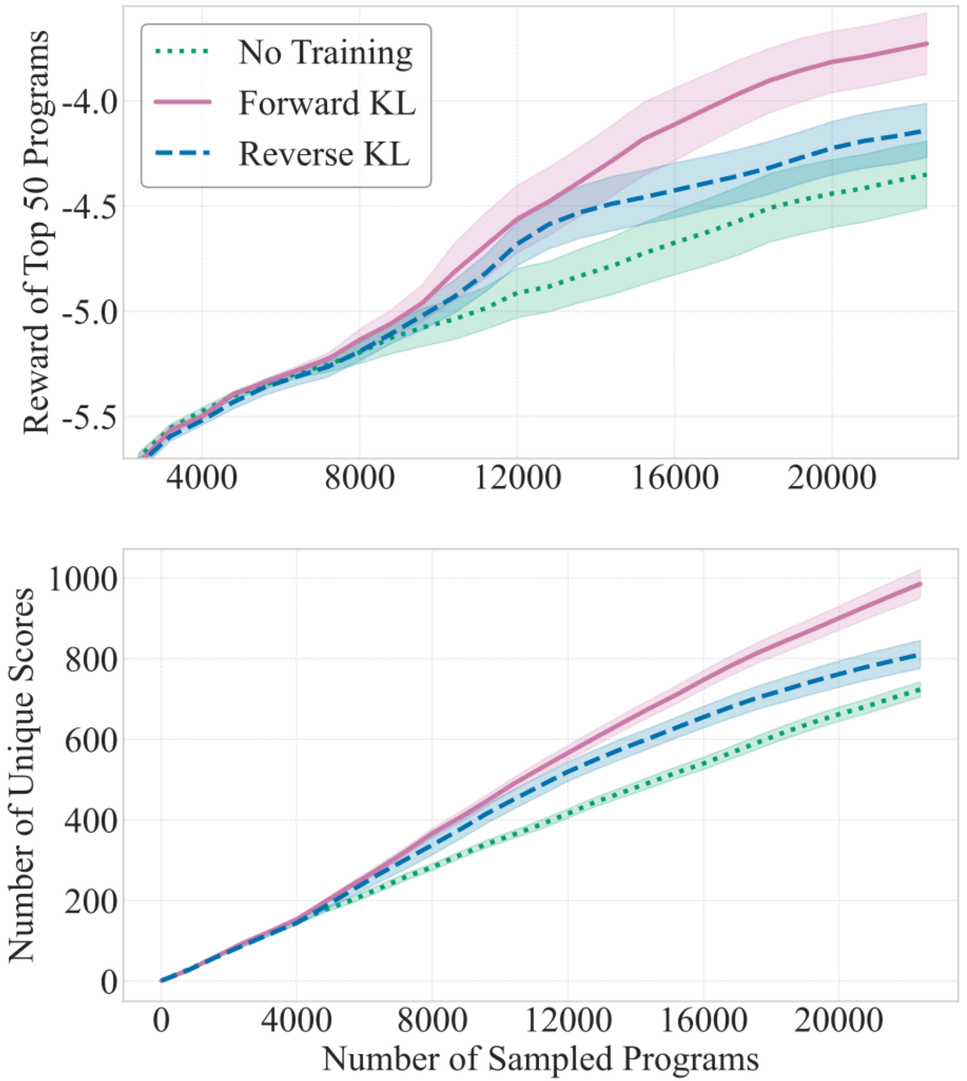
Forward KL: Mass covering





Case study: Problem 2

Forward vs Reverse KL



From Surina et al., COLM, 2025

Summary

FunSearch: evolutionary algorithm to search good Python functions

- Output understandable by humans
- Safeguard against hallucinations by asking Python code that we can automatically run to verify the solution

EvoTune: add RL to FunSearch method

- More diversity of outputs
- Better results

Questions?