AI in Mathematics
Lecture 9
NLP and LLMs

Bar-Ilan University
Nebius Academy | Stevens Institute of Technology

May 20, 2025

About This Course

~~1 week: Intro~~

~~2 weeks: Classic ML~~

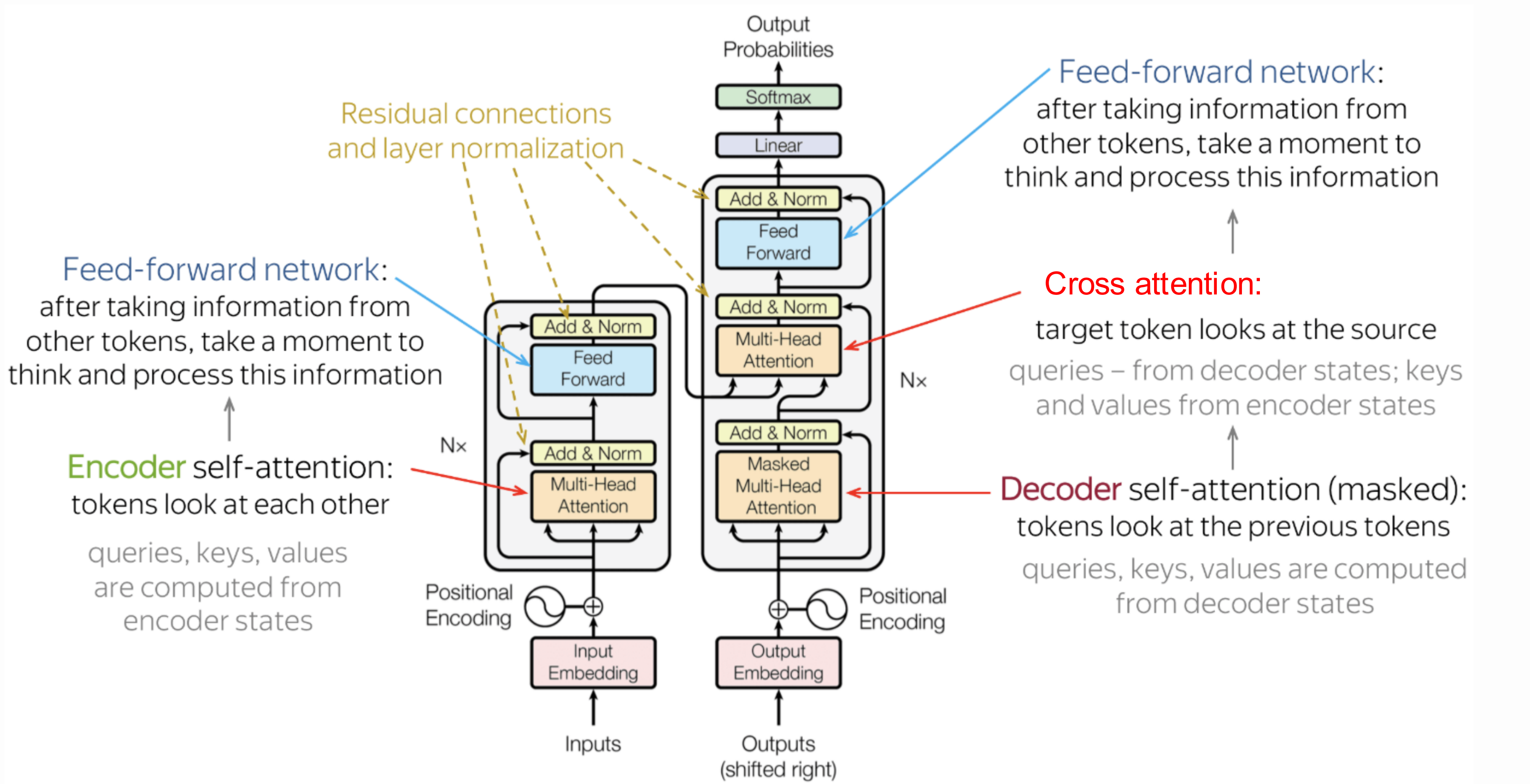
~~2 weeks: Deep Learning in Mathematics~~

4 weeks: Math as an NLP problem (LLMs etc.)

3 weeks: Reinforcement Learning (RL) in Math

1 week: Advanced AI topics or Project Presentations

Transformer



Encoder-only vs Decoder-only Models

Encoder only

Example:
BERT(Bidirectional Encoder Representations from **Transformers**)

• Sentiment analysis
• Named entity recognition
• Question answering (extractive)
• Sentence similarity

Decoder only

Example:
GPT (Generative Pre-trained **Transformer**)

• Text generation
• Code completion
• Chatbots
• Story writing

BERT

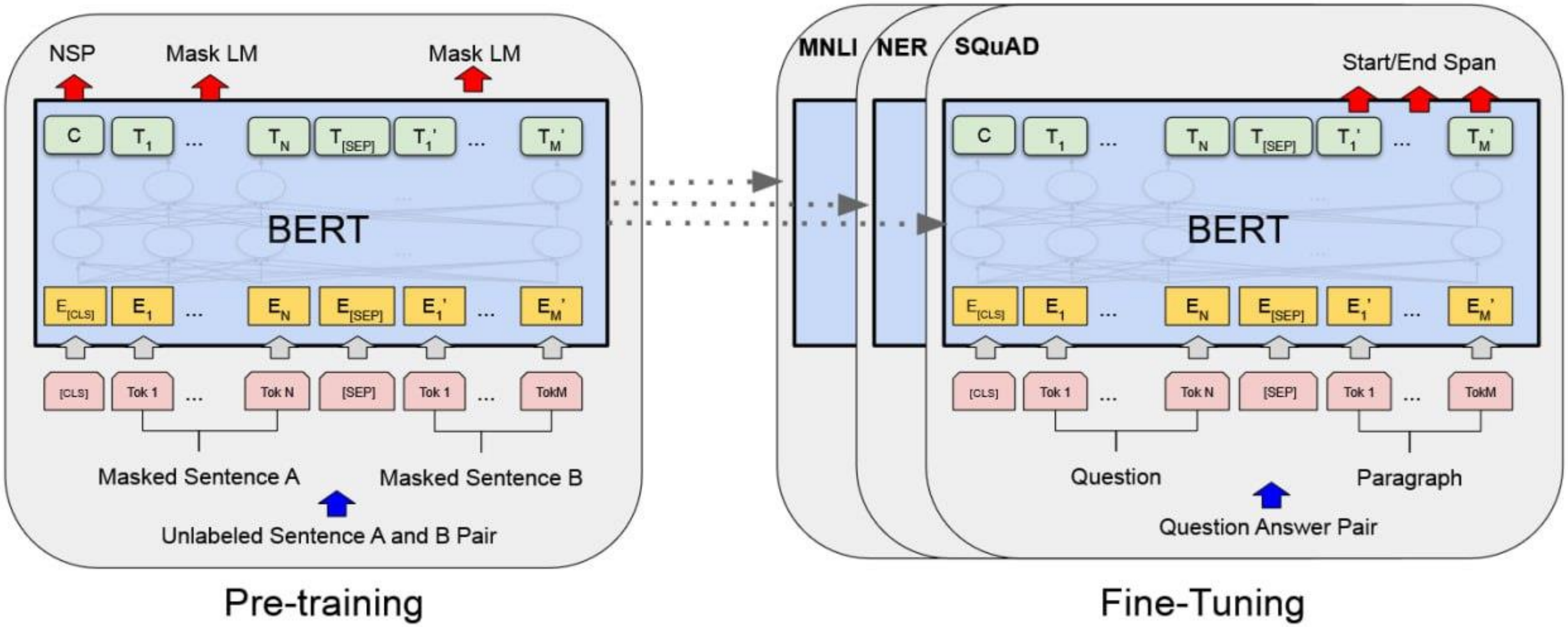


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

Pre-trained models

LLMs like **GPT-4o**, **LLaMA**, Claude, Gemini, etc.
have broad knowledge, but to solve **specific tasks**, we need to guide them appropriately:

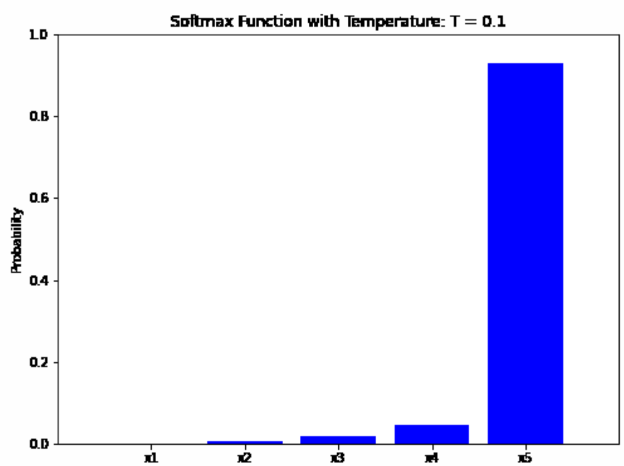
📜 **Prompting** – e.g., “Write code to solve…”

🛠 **Fine-tuning** – train the model on task-specific examples

Temperature

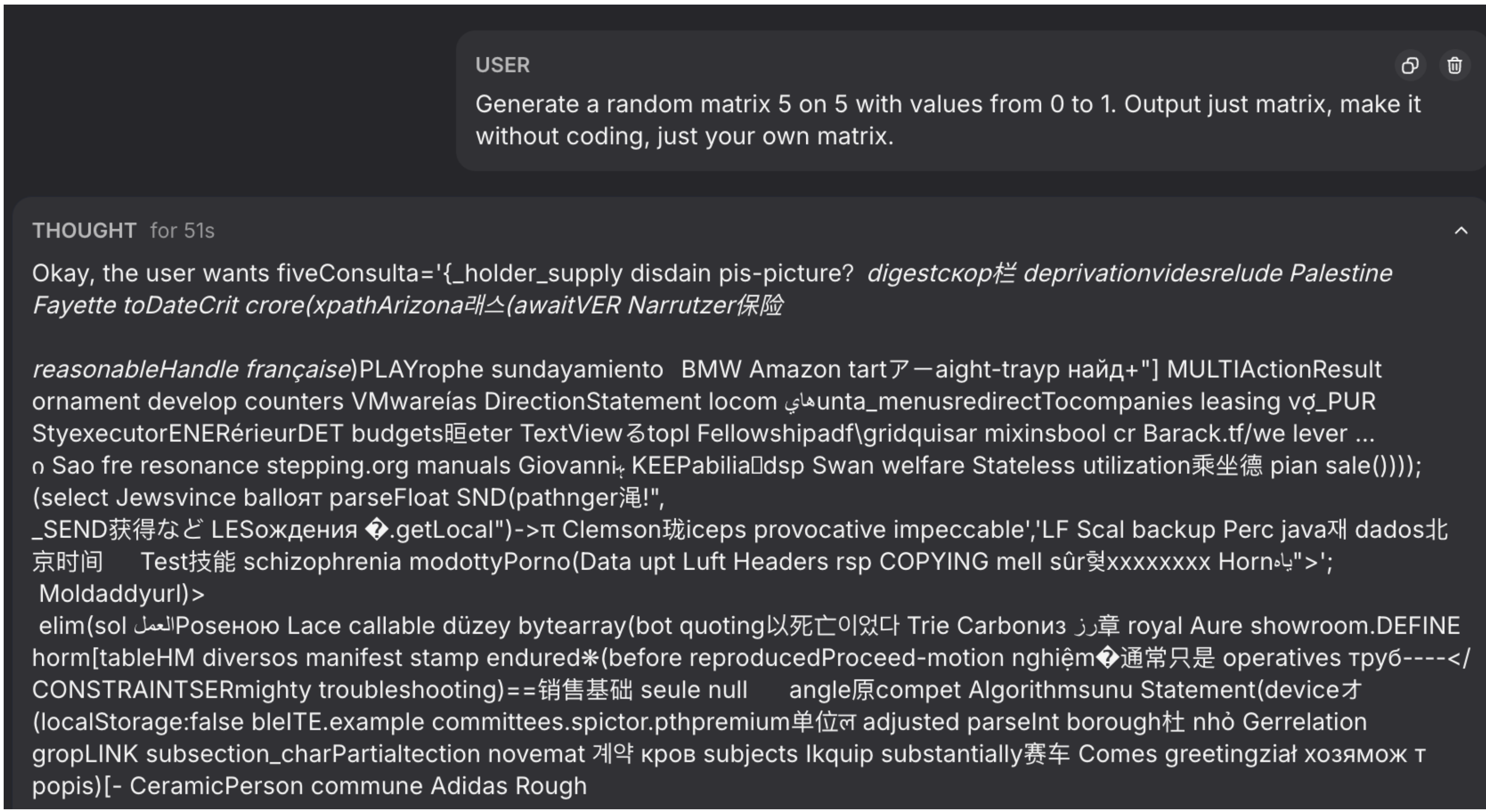
$$Softmax((x_1, x_2, \dots, x_n), t) = \left(\frac{e^{\frac{x_1}{t}}}{\sum_{i=1}^{n} e^{\frac{x_i}{t}}}, \dots, \frac{e^{\frac{x_n}{t}}}{\sum_{i=1}^{n} e^{\frac{x_i}{t}}} \right)$$

In theory, softmax with temperature → 0 approaches a one-hot distribution over the argmax. However, temperature = 0 is **not used in practice**, as it causes numerical instability. Instead, argmax is implemented explicitly as a special case.



Temperature

Temperature = 2 in real life model:



Even temperature = 1 can cause such problems.

Temperature

Temperature 0.6 for real life model:

THOUGHT for 10s

Alright, the user has asked me to generate a 5x5 matrix with values between 0 and 1 without using any coding. They want me to create it on my own. Let me think about how to approach this.

First, I need to make sure I understand the request correctly. They don't want me to write code or use any programming languages. Instead, they want a matrix that I come up with manually. Since it's random, I should generate numbers in that range without any specific pattern.

Hmm, how can I create such a matrix? Well, I can think of numbers between 0 and 1, maybe using two decimal places for simplicity. I should create 25 unique numbers, but since it's random, a few repetitions might be okay.

Wait, the user mentioned "without any agents." Does that mean I shouldn't use external tools or algorithms? Probably, just to ensure it's truly my own creation. I'll have to rely on my own ability to generate random numbers mentally.

Let me start filling in the matrix row by row. For the first row: 0.12, 0.45, maybe 0.67, then 0.89, and 0.34. That seems random enough. Second row: 0.56, 0.23, 0.91, 0.05, 0.78. Checking for duplicates so far, looks okay.

Third row: 0.90, 0.01, 0.35, 0.72, 0.49. Fourth row: 0.63, 0.28, 0.50, 0.82, 0.19. Fifth row: 0.85, 0.31, 0.46, 0.69, 0.15. Let me review each number to ensure they're between 0 and 1 and no obvious patterns.

I think this works. The numbers are spread out, not following any sequence. I don't see any numbers outside the 0-1 range. Alright, this should meet the user's requirements. They just need the matrix displayed without any code involved.

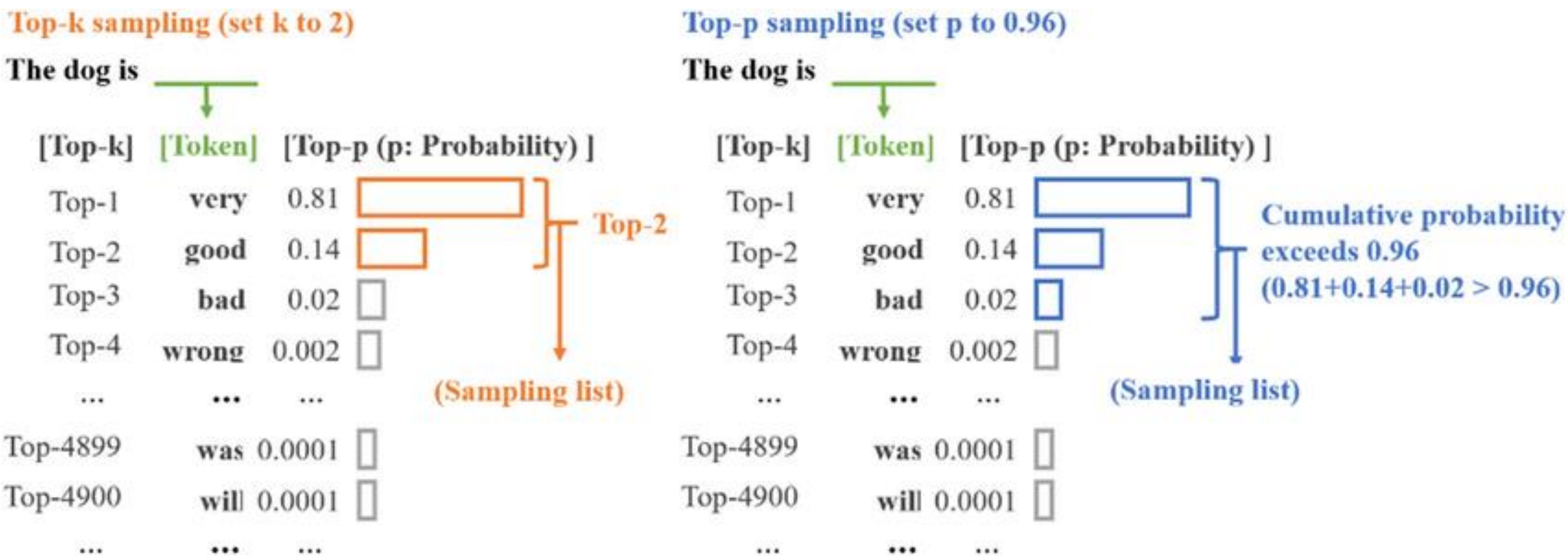
AI ASSISTANT 576 tokens

Here is a randomly generated 5x5 matrix with values between 0 and 1:

```
0.12 0.45 0.67 0.89 0.34
0.56 0.23 0.91 0.05 0.78
0.90 0.01 0.35 0.72 0.49
0.63 0.28 0.50 0.82 0.19
0.85 0.31 0.46 0.69 0.15
```

Regenerate 587 ms | 14.3 s | 40.5 t/s Add as example

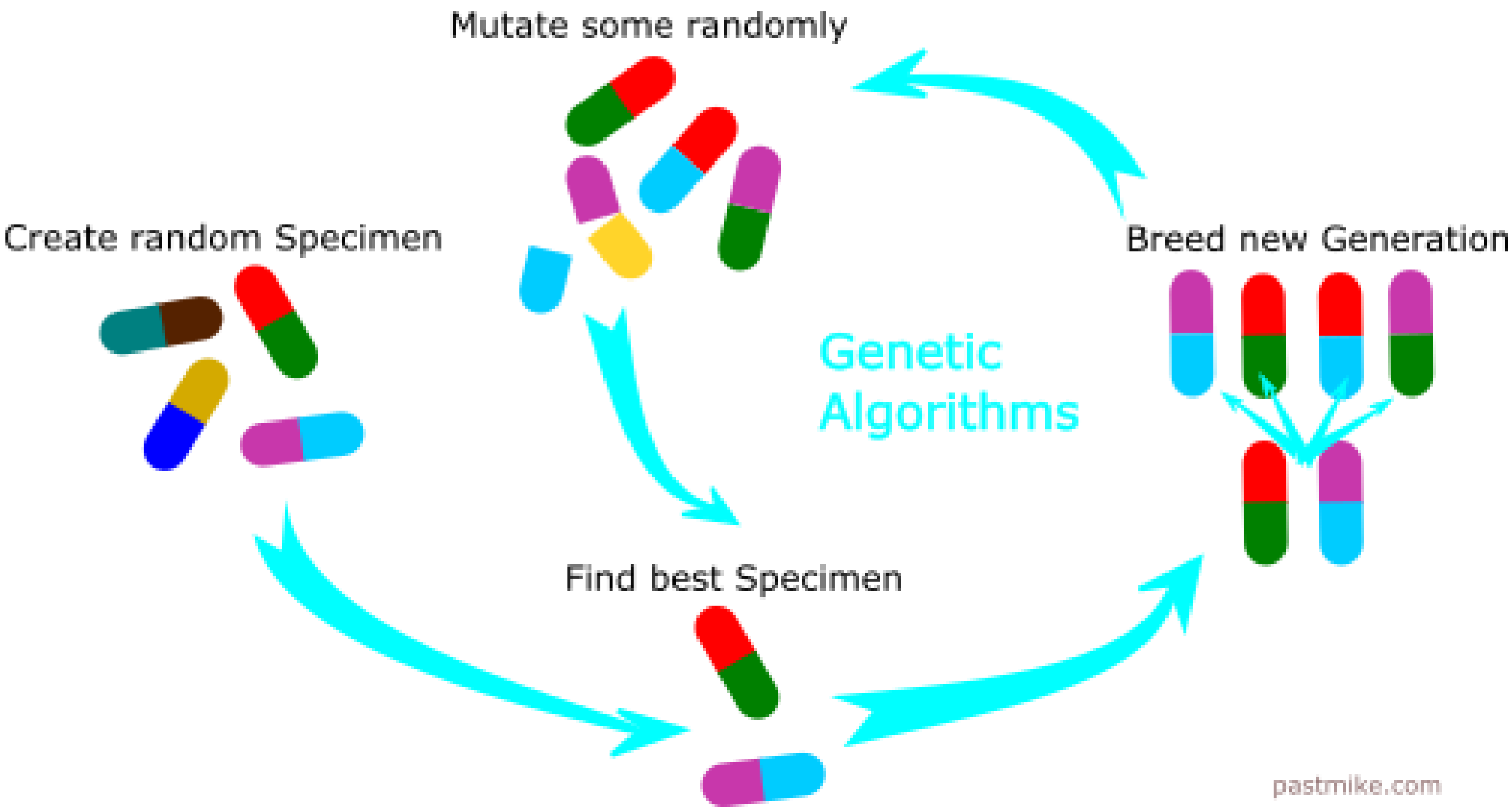
Top-k and top-p



Models often use a **hyperparameters** to control the possible pool of tokens in sampling.

Evolutionary algorithm

”Survival of the Fittest”

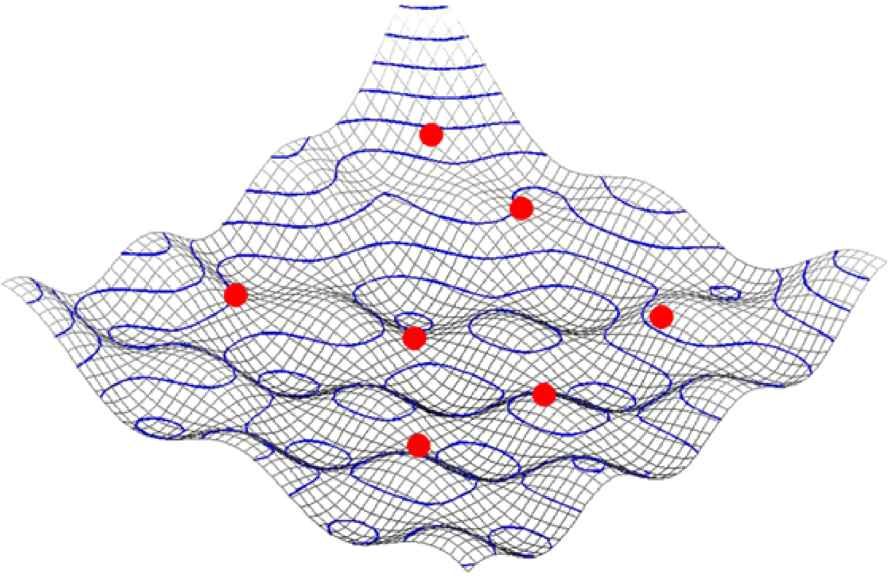


Evolutionary algorithm

For example:

Finding a minimum of a function via following algorithm:

Select a population of points. Try to move them a little and select new state for the smallest value of function.



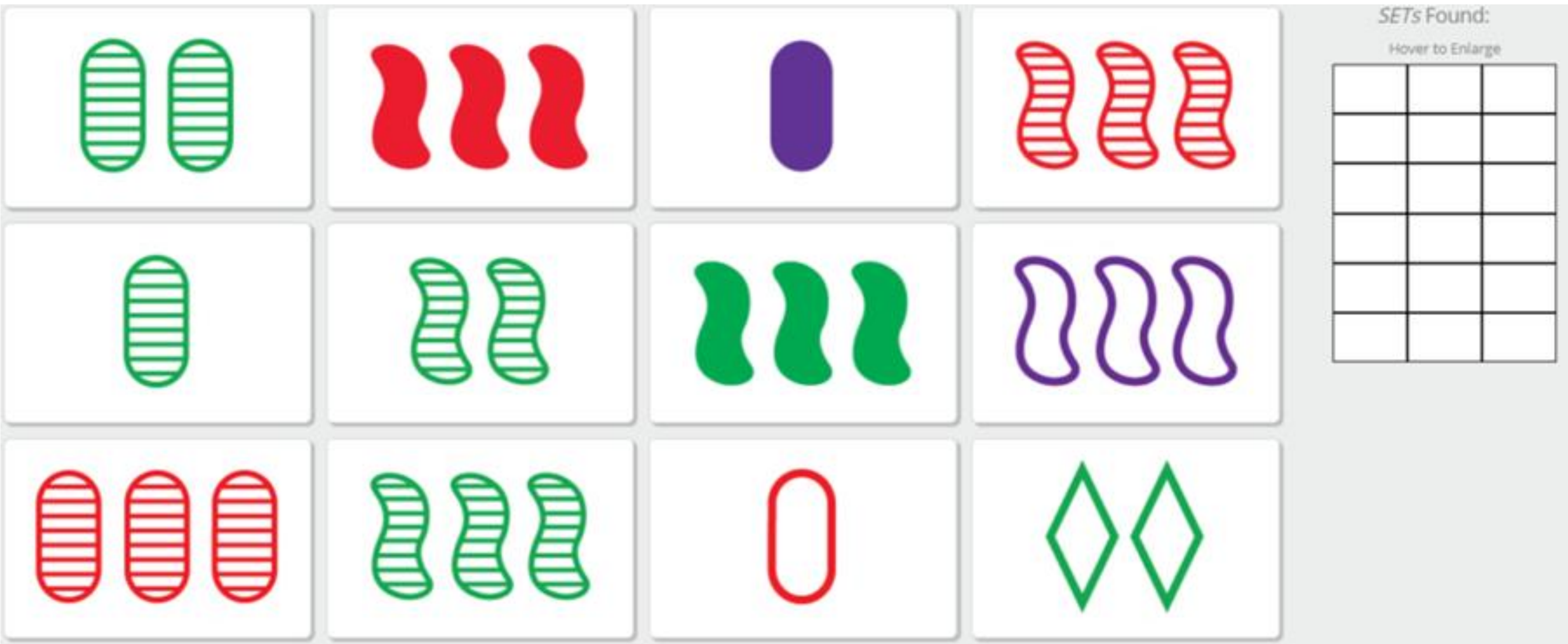
FunSearch

FunSearch is a **functional search algorithm** that has shown success on certain **mathematical problems**.

📌 For example: the **capset problem**, which we introduced in the first lecture.

👉 We will now explore it in more detail.

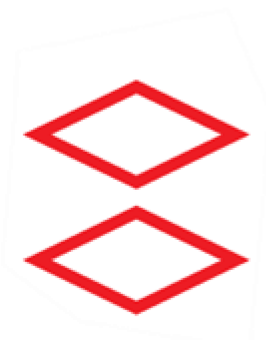
Reminder: The SET Game



Reminder: The SET Game

What mathematical structure do the cards correspond to?

Each card represents a point in $\mathbb{Z}_3^4$ (a four-dimensional vector space over $\mathbb{Z}_3$). A ”set” is a triplet of cards whose sum in $\mathbb{Z}_3^4$ is the zero vector



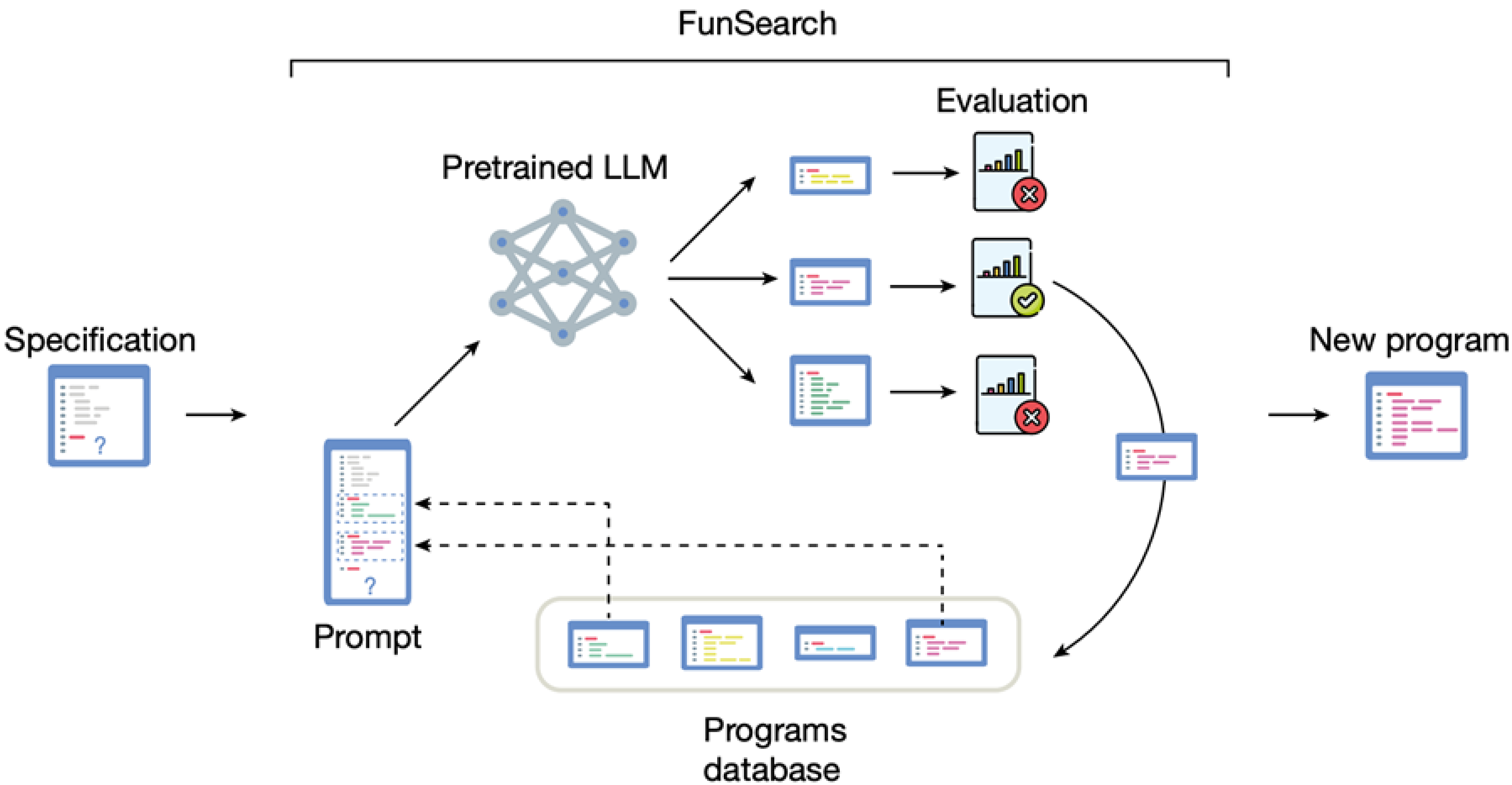
(diamond, 2, red,open)

AI improved the results!

Find the largest possible subset of $\mathbb{Z}_3^n$, such that sum of any triplet doesn't equal to zero.

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

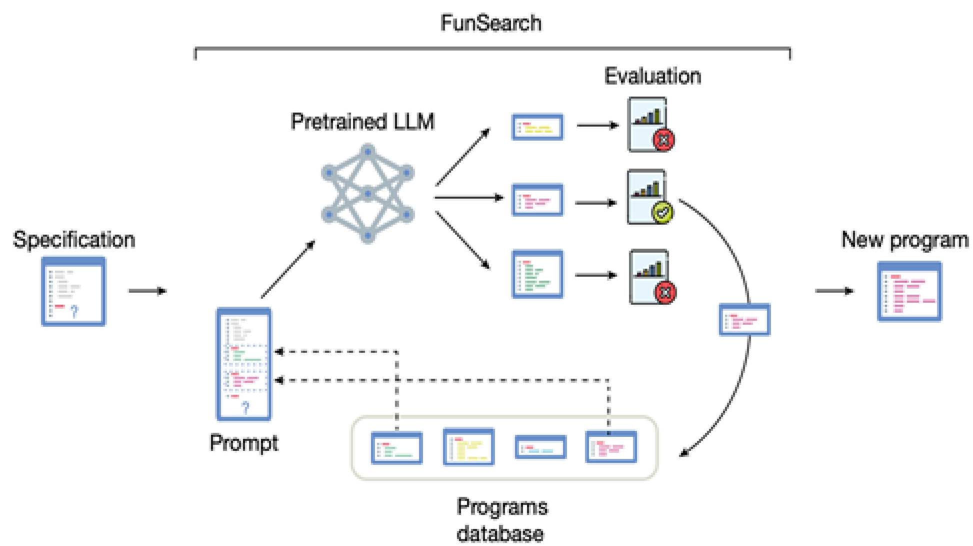
FunSearch paradigm



FunSearch

Goal:
Generate functions that construct large capsets.

Ensure the generated functions exhibit a **generalizable structure**, so the model learns **underlying patterns** rather than merely memorizing solutions.



Important Idea

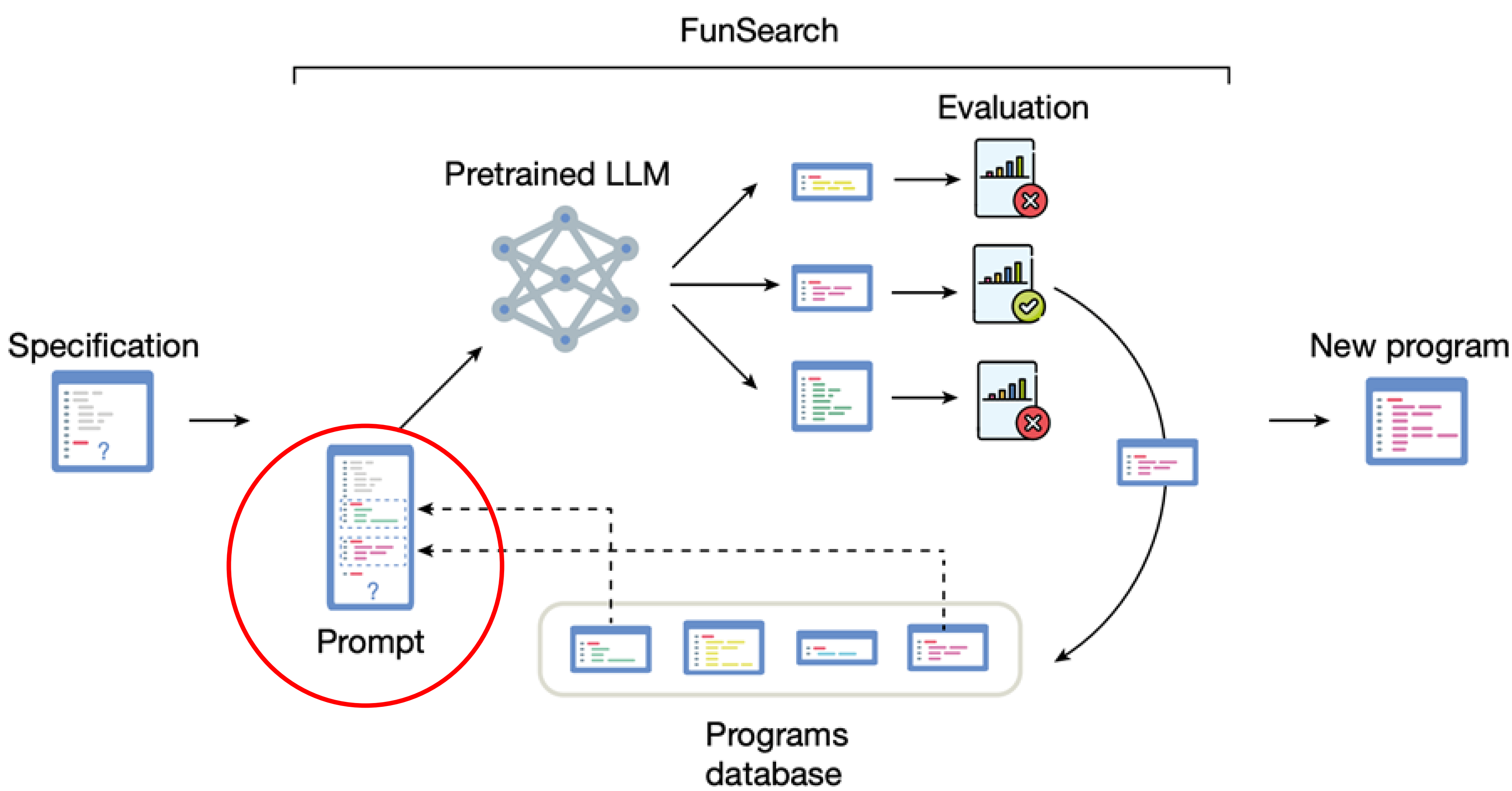
Instead of generating a full selection function for cap sets, the LLM learns a **scoring function**

$$f : \mathbb{Z}_3^n \to \mathbb{R}$$

The **set selection** is done via **greedy search** using this scoring function.

Motivation:
The goal is to *"evolve only the part governing the critical program logic."*

FunSearch paradigm



Prompt

In the upcoming practice session, we'll explore how to prompt LLMs effectively.

Example from the FunSearch paper:

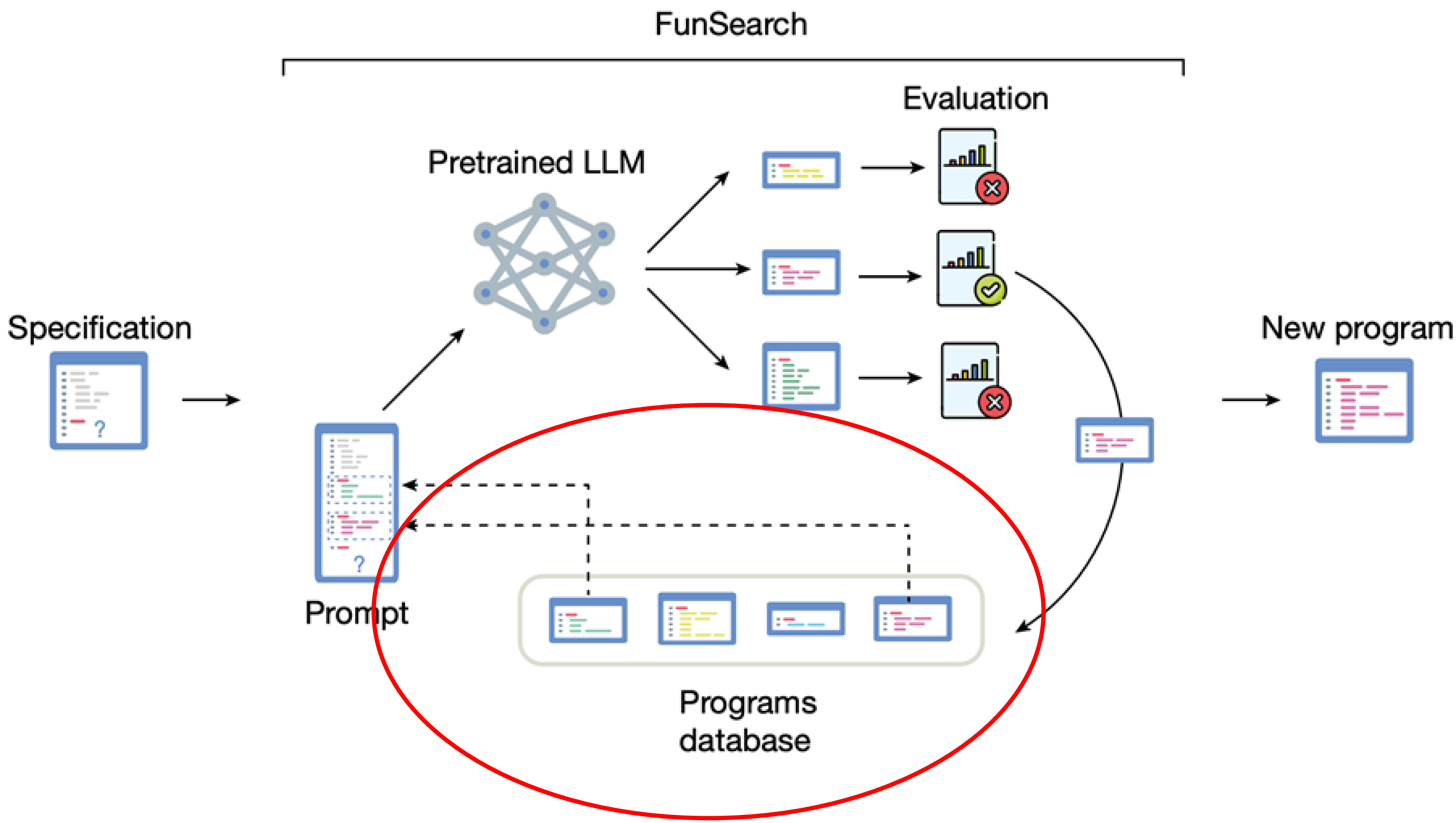
```python
"""Finds large cap sets."""
import numpy as np
import utils_capset

def priority_v0(element, n):
  """Returns the priority with which we want to add `element` to the cap set."""
  #######
  # Code from lowest-scoring sampled program.
  return ...
  #######

def priority_v1(element, n):
  """Improved version of `priority_v0`."""
  #######
  # Code from highest-scoring sampled program.
  return ...
  #######

def priority_v2(element, n):
  """Improved version of `priority_v1`."""
```

Overfitting

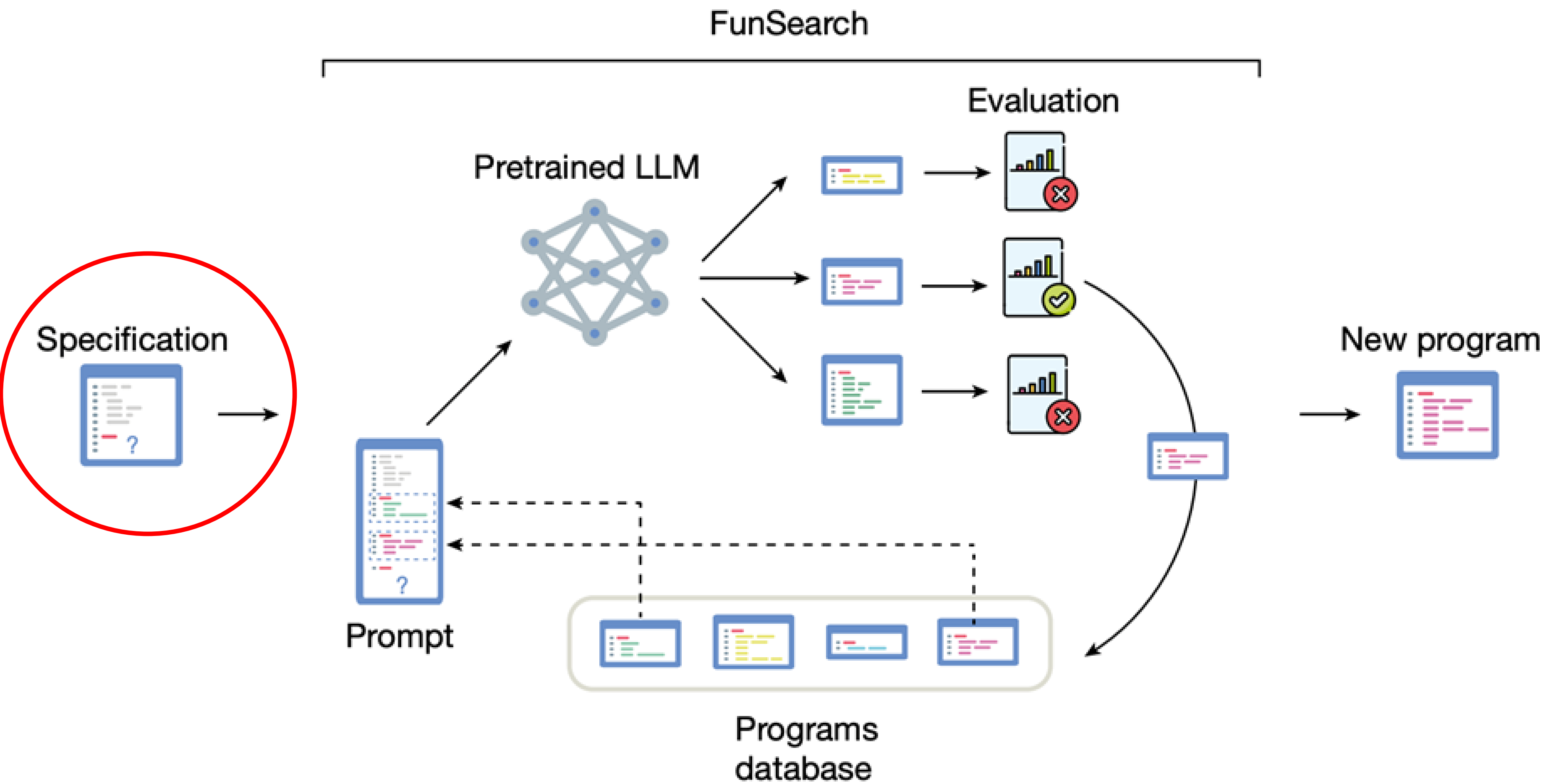


Avoiding Overfitting During Search

To reduce overfitting, authors used the following strategy:

🔀 **Split the database** into k parts every 4 hours
🕛 Each part evolves **independently** for 4 hours
♻️ At the start of a new epoch, **programs are reshuffled**
🔁 Evolutionary steps (mutations/selection) are applied **after each 4-hour cycle**

FunSearch paradigm



Specification

Inputs to the Search:

Reference function: $Solve$
Provides a starting solution — a function of n that returns a candidate capset.

Evaluation function: $Evaluate$
Takes a candidate capset as an input. Returns the **size** of the solution if it is a valid capset; otherwise, returns zero.

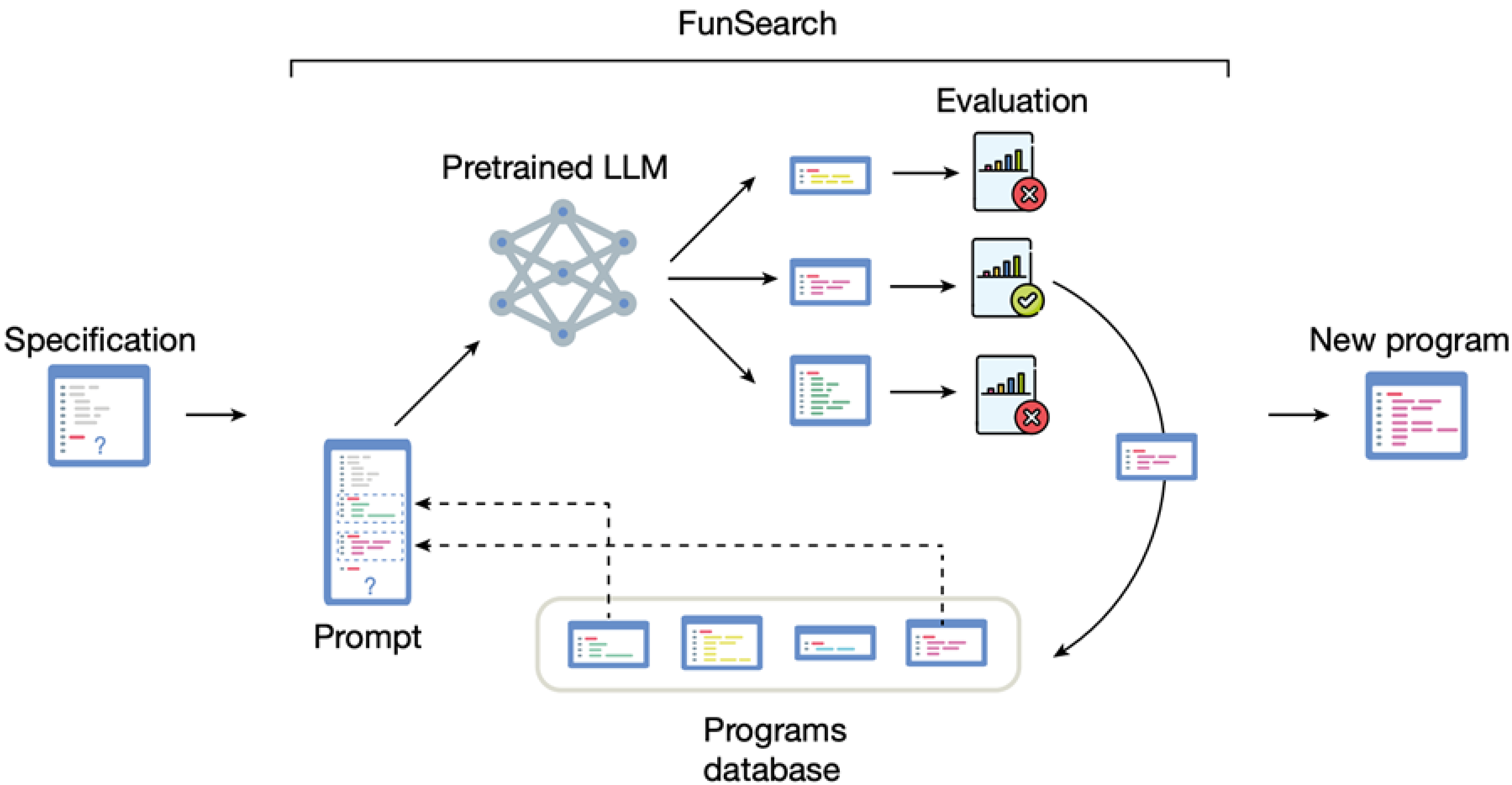
Specification

```
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
```

FunSearch paradigm



Understanding a growth rate of a Capset

The central question in capset research:

What is $\boldsymbol{\gamma = \sup_n c_n^{\frac{1}{n}}}$, where $\boldsymbol{c_n}$ is the size of a largest capset in $\mathbb{Z}_3^n$?

It is easy to see that $2^n \leq c_n < 3^n$, right?

Analyzing a cap set constructed by **FunSearch**, one of the authors improved the best-known **lower bound**:
$2.2202 < \gamma$ (previousy $2.2180 < \gamma$).
Upper bound remains $\gamma < 2.756$.

Some discussion

What Makes a Task Suitable for FunSearch?

According to the authors, success requires:

✅ An **efficient evaluation function**
✅ A **rich scoring signal**, not just binary feedback
✅ Only a **small part of the algorithm** is evolved

Critique Highlights an Additional Requirement:
There must be a rich space of candidate functions, with meaningful variation in performance — some solutions better, some worse.

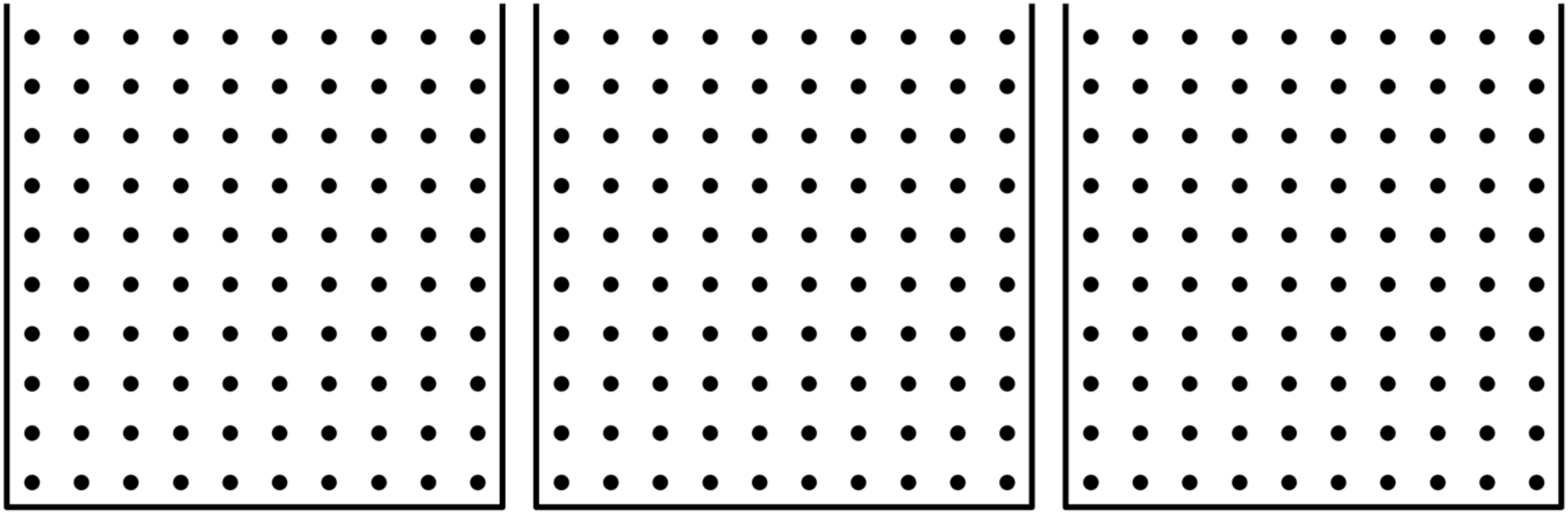
Other problems tackled by FunSearch

📦 **Online Bin Packing**
Combinatorial optimization under online constraints.

📡 **Shannon Capacity of a Graph**
At the intersection of combinatorics and information theory.

🟦 **Corners Problem**
A classic in combinatorics, involving geometric patterns in grids.

Online bin packing



Elements arrive **one by one**, and for each incoming element you must **immediately decide** where to place it.
❗**Decisions are irreversible** — no changes allowed after placement.

Heuristics

Classical:

first-fit – number the bins and put the object in the first bin where it fits.

best-fit – put the object into the fullest bin where it fits.

FunSearch:

```
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
```

Results

	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%**

Fraction of excess bins (lower is better) for various bin packing heuristics on the OR and Weibull datasets. FunSearch outperforms first fit and best fit across problems and instance sizes.

Comment:

FunSearch was trained on a dataset generated from OR1

Resource usage

Reproducing the results:

Compute:

- 15× StarCoder-15B on A100 40GB GPUs
- 5× CPU servers (32 evaluators each)
- Runtime: 2 days

Estimated Cost:

$800 – $1400 on Google Cloud

Energy Usage:

250 – 500 kWh