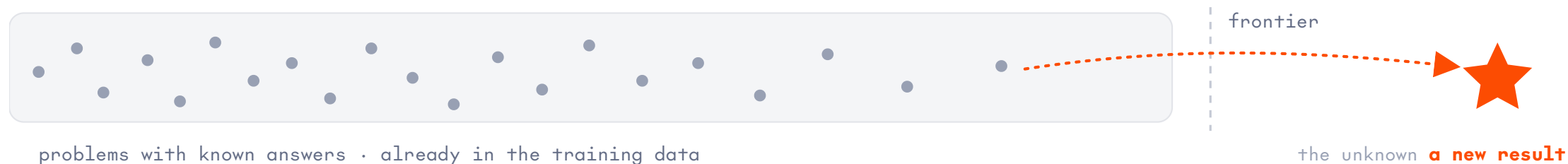


RESEARCH • AGENTS THAT DISCOVER

Learning to Discover at **Test Time**.

— **Federico Bianchi** Staff Scientist, Together AI



Learning to Discover at Test Time. ICML 2026.

Yuksekgonul*, Kocejda*, Li*, **Bianchi***, McCaleb, Wang, Kautz, Choi, Zou, Guestrin & Sun.

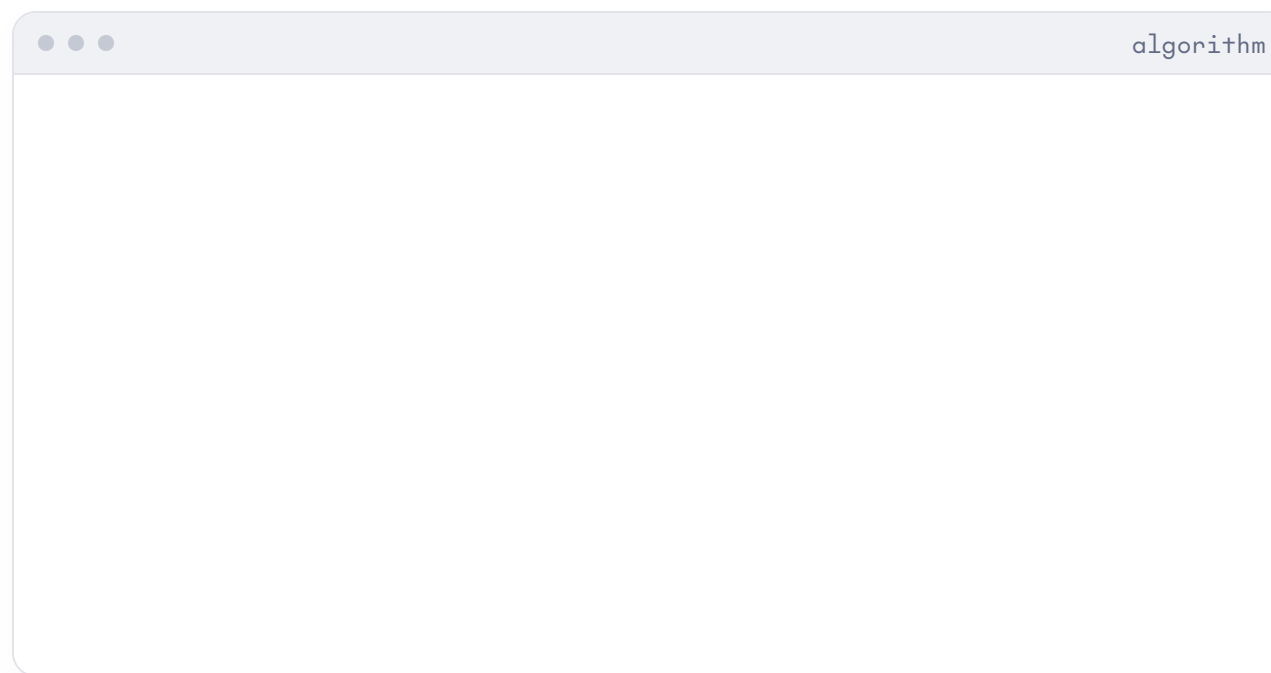
* equal contribution · Stanford University · NVIDIA · Astera Institute · UC San Diego · Together AI



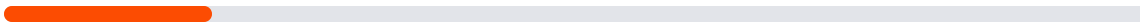
Federico Bianchi

Staff Scientist at Together AI, previously OpenEvidence and Stanford. I work on **self-improving agents** and **scientific discovery**.

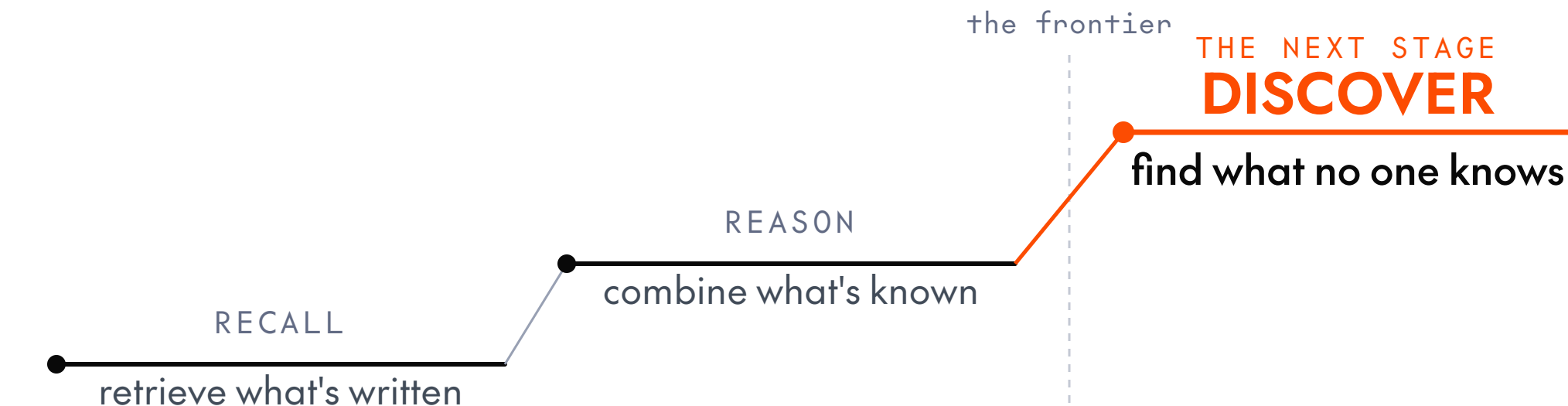
Today's talk is about **algorithmic discovery**.



Improvement loop

better 

The community has been moving towards **discovery**.



Automated scientific discovery.

Machines are **already discovering**.

 Google DeepMind 

May 14, 2025 Science

AlphaEvolve: A Gemini-powered coding agent for designing advanced algorithms

AlphaEvolve team

sakana.ai

ShinkaEvolve: Evolving New Algorithms with LLMs, Orders of Magnitude More Efficiently

September 25, 2025

AlphaEvolve found a faster matrix multiply.

★ A NEW DISCOVERY

● ● ● multiply_4x4.py

```
def multiply(A, B): # Strassen, 1969
    m1 = (a11 + a22) * (b11 + b22)
    m2 = (a21 + a22) * b11
    m3 = a11 * (b12 - b22)
    ... # 7 products per 2x2 block
    # 4x4 = 7 x 7 = 49
```



● ● ● multiply_4x4.py

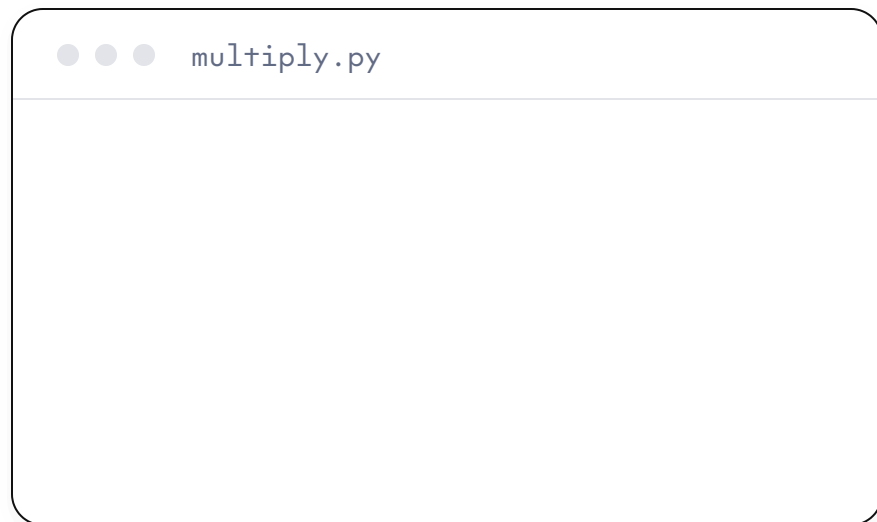
```
def multiply(A, B): # AlphaEvolve, 2025
    m1 = ( $\frac{1}{2}a_{11} + \frac{1}{2}a_{31} + \dots$ ) * (b12 + ...)
    m2 = (a12 - a13 + ...) * ( $\frac{1}{2}b_{21} - \dots$ )
    m3 = (a11 +  $\frac{1}{2}i \cdot a_{41} \dots$ ) * (b13 + ...)
    ... # 48 products · 4x4 complex-valued matrices
    # first improvement in 56 years
```

49 multiplications

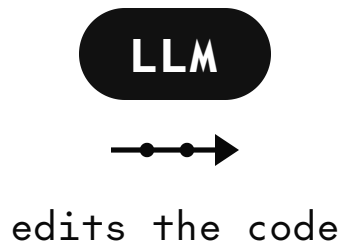
48 multiplications

An LLM **edits code** and keeps what scores better.

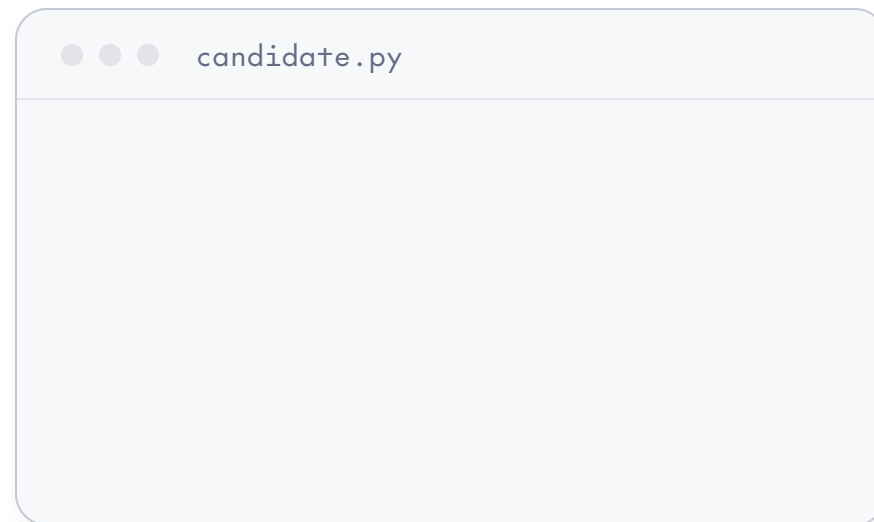
THE PROBLEM



49 multiplications

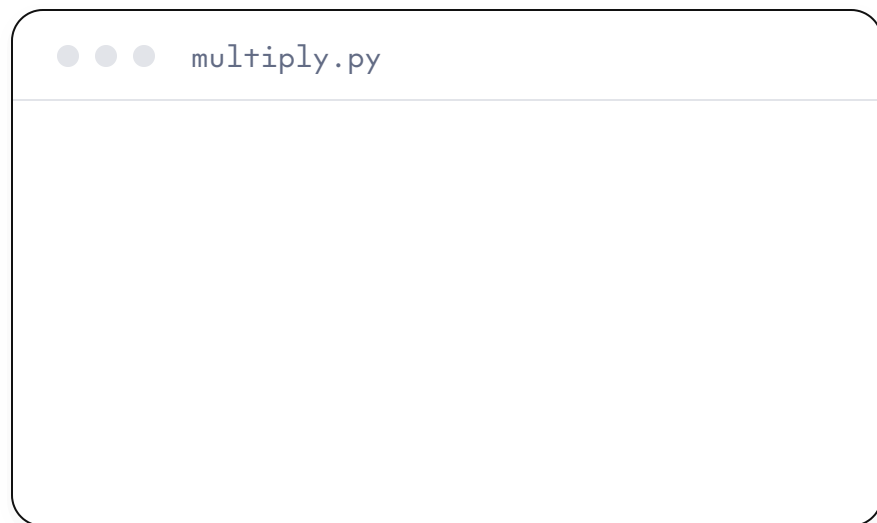


CANDIDATE

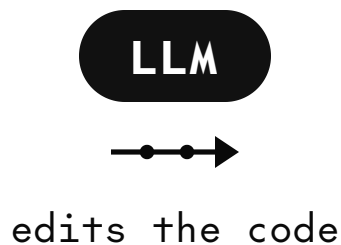


An LLM **edits code** and keeps what scores better.

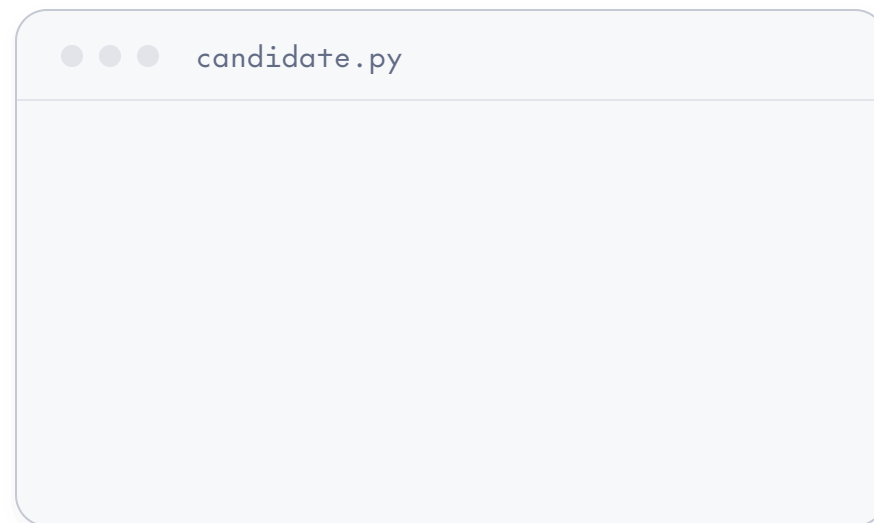
THE PROBLEM



49 multiplications



CANDIDATE



What counts as a discovery.

$$R(s) > R(s_{\text{sota}})$$

What counts as a discovery.

$$R(s) > R(s_{\text{sota}})$$

s the model's solution

What counts as a discovery.

$$R(s) > R(s_{\text{sota}})$$

s the model's solution

R the verifier $R : \mathcal{D} \rightarrow \mathbb{R}$

What counts as a discovery.

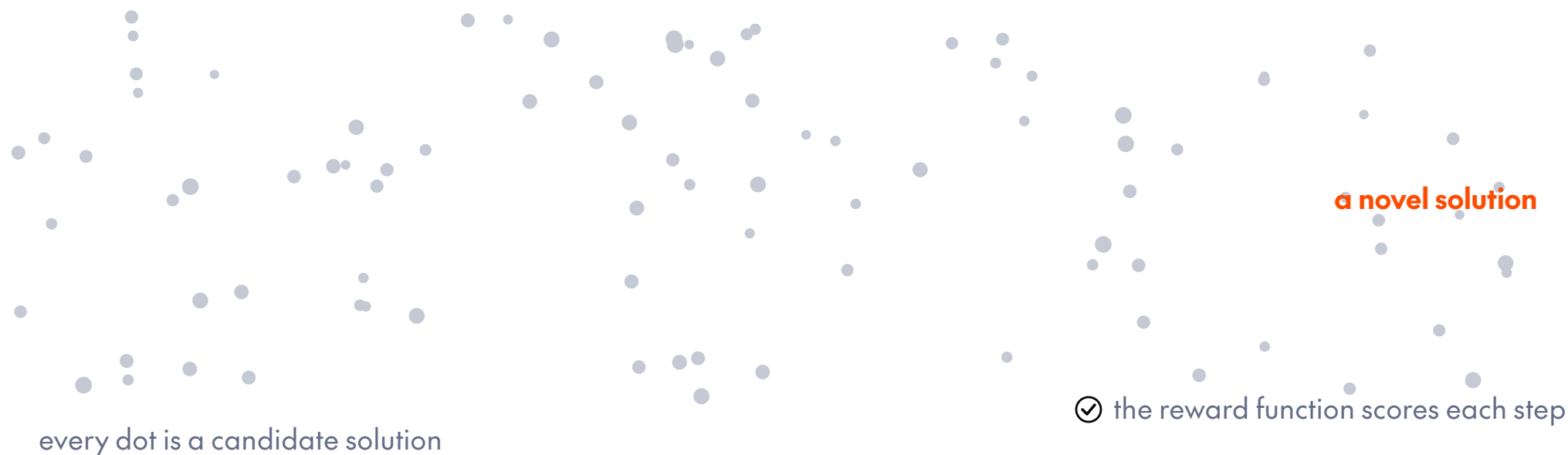
$$R(s) > R(s_{\text{sota}})$$

s the model's solution

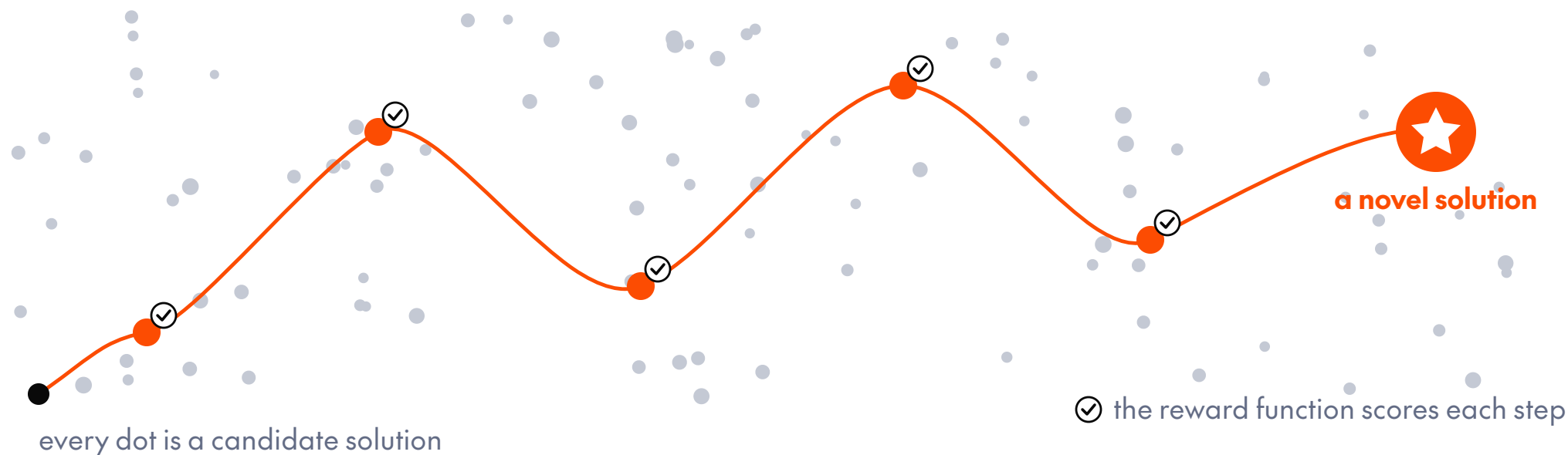
R the verifier $R : \mathcal{D} \rightarrow \mathbb{R}$

s_{sota} the best known solution

A huge space of candidates: we need guided search and a reward function.



A huge space of candidates: we need guided search and a reward function.



Some of **our work** in this space.

Some of **our work** in this space.

• NATURE • 2025

Text+Grad

Optimizing systems
with natural-language
feedback.

Some of **our work** in this space.

• NATURE • 2025

Text+Grad

Optimizing systems
with natural-language
feedback.

• NATURE BIOTECHNOLOGY • 2025

Agents4Science

Testing whether agents
can do scientific work
end to end.

Some of **our work** in this space.

• NATURE • 2025

Text+Grad

Optimizing systems
with natural-language
feedback.

• NATURE BIOTECHNOLOGY • 2025

Agents4Science

Testing whether agents
can do scientific work
end to end.

• ARXIV • 2026

EinsteinArena

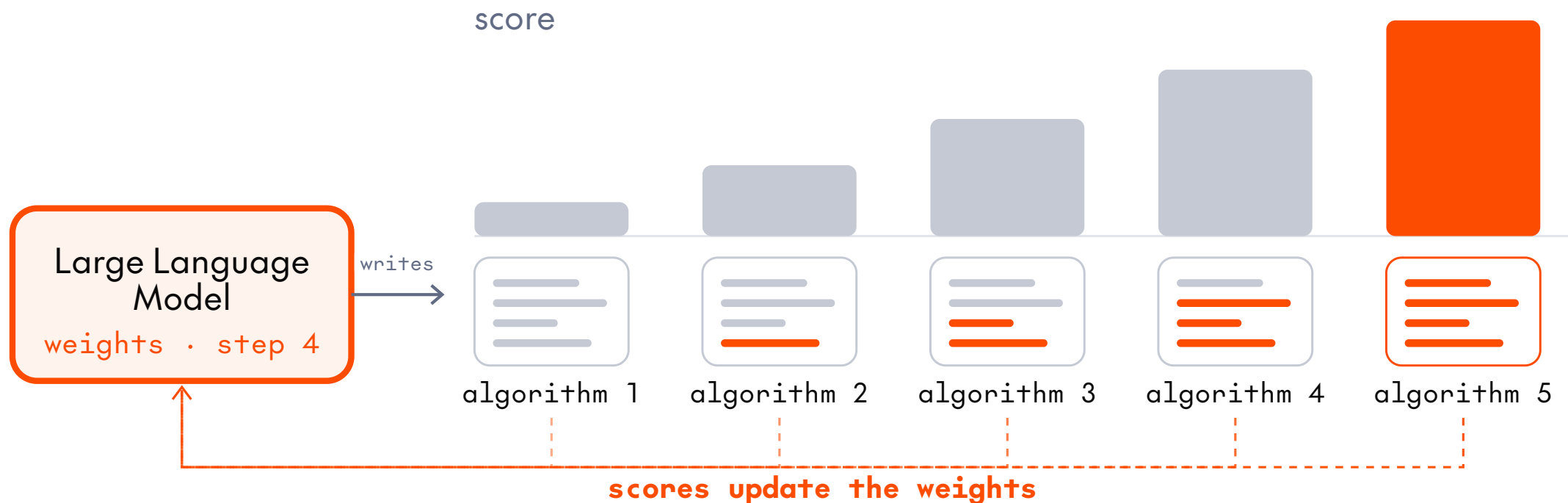
Exploring agent
collaboration on
verifiable math
problems.

TTT-Discover

What if the model kept learning while it worked?

A model that **learns** at test time writes algorithms that keep getting better.

A model that **learns** at test time writes algorithms that keep getting better.



A recipe for **discovery**.

A recipe for **discovery**.

1

Reuse the best attempts

Reuse promising states to
extend the horizon.

A recipe for **discovery**.

1

Reuse the best attempts

Reuse promising states to
extend the horizon.

2

Train on the one
problem

RL on one problem at test
time.

A recipe for **discovery**.

1

Reuse the best attempts

Reuse promising states to extend the horizon.

2

Train on the one problem

RL on one problem at test time.

3

Reward the best sample

Optimize for the maximum.

New records on famous problems.

New records on famous problems.



Erdős minimum
overlap

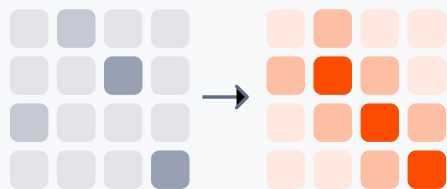
Open since 1955

All with one open model **gpt-oss-120b** | en.wikipedia.org/wiki/Minimum_overlap_problem

New records on famous problems.



Erdős minimum
overlap
Open since 1955

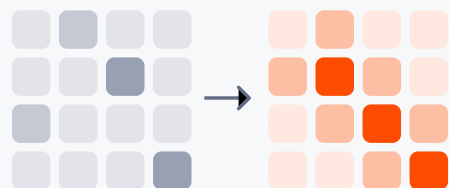


Single-cell denoising
Beats MAGIC

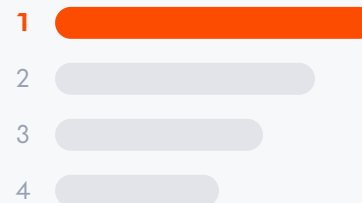
New records on famous problems.



Erdős minimum
overlap
Open since 1955



Single-cell denoising
Beats MAGIC

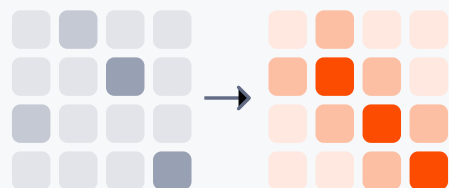


Algorithm
engineering
1st in two AtCoder
contests

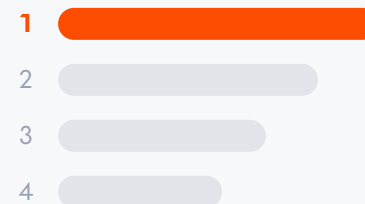
New records on famous problems.



Erdős minimum
overlap
Open since 1955



Single-cell denoising
Beats MAGIC



Algorithm
engineering
1st in two AtCoder
contests



GPU kernels
up to 2× faster than the
record (A100)

TODAY

One kernel's story: from a **single prompt** to a machine that **discovers**.

This talk tells the paper as one incremental story rather than in paper order. Some numbers come from different experiments and are combined for coherence; the impact is minimal, but see the paper for exact details.

Many algorithms we could discover.

Many algorithms we could discover.



GPU KERNEL

TriMul



trimul.py

```
@triton.jit
def trimul(x, w, out, BLOCK):
    pid = tl.program_id(0)
    offs = tl.arange(0, BLOCK)
    offs += pid * BLOCK
    a = tl.load(x + offs)
    acc = tl.dot(a, w)
    tl.store(out + offs, acc)
```

- minimize μ s

Many algorithms we could **discover**.



GPU KERNEL

TriMul

trimul.py

```
@triton.jit
def trimul(x, w, out, BLOCK):
    pid = tl.program_id(0)
    offs = tl.arange(0, BLOCK)
    offs += pid * BLOCK
    a = tl.load(x + offs)
    acc = tl.dot(a, w)
    tl.store(out + offs, acc)
```

- minimize μ s



SCHEDULING

Production scheduling

schedule.py

```
def local_search(order):
    best = score(order)
    while time_left():
        i, j = pick_pair(order)
        swap(order, i, j)
        if score(order) > best:
            best = score(order)
        else: swap(order, i, j)
```

- maximize score

Many algorithms we could discover.



TriMul

trimul.py

```
@triton.jit
def trimul(x, w, out, BLOCK):
    pid = tl.program_id(0)
    offs = tl.arange(0, BLOCK)
    offs += pid * BLOCK
    a = tl.load(x + offs)
    acc = tl.dot(a, w)
    tl.store(out + offs, acc)
```

- minimize μ s



Production scheduling

schedule.py

```
def local_search(order):
    best = score(order)
    while time_left():
        i, j = pick_pair(order)
        swap(order, i, j)
        if score(order) > best:
            best = score(order)
        else: swap(order, i, j)
```

- maximize score



Portfolio rebalancing

rebalance.py

```
def rebalance(returns, w_prev):
    mu, cov = estimate(returns)
    w = solve_qp(mu, cov,
                 risk_cap=0.12,
                 turnover=0.05)
    trades = w - w_prev
    execute(trades)
    return sharpe(w, returns)
```

- maximize Sharpe

Many algorithms we could **discover**.



TriMul

trimul.py

```
@triton.jit
def trimul(x, w, out, BLOCK):
    pid = tl.program_id(0)
    offs = tl.arange(0, BLOCK)
    offs += pid * BLOCK
    a = tl.load(x + offs)
    acc = tl.dot(a, w)
    tl.store(out + offs, acc)
```

- minimize μs



Production scheduling

schedule.py

```
def local_search(order):
    best = score(order)
    while time_left():
        i, j = pick_pair(order)
        swap(order, i, j)
        if score(order) > best:
            best = score(order)
        else: swap(order, i, j)
```

- maximize score



Portfolio rebalancing

rebalance.py

```
def rebalance(returns, w_prev):
    mu, cov = estimate(returns)
    w = solve_qp(mu, cov,
                 risk_cap=0.12,
                 turnover=0.05)
    trades = w - w_prev
    execute(trades)
    return sharpe(w, returns)
```

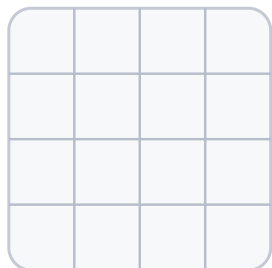
- maximize Sharpe

A kernel is a math operation **implemented** for a specific chip.

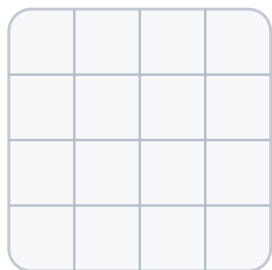
a math operation implemented for a specific chip

A kernel is a math operation **implemented** for a specific chip.

$$C = A \times B$$



A

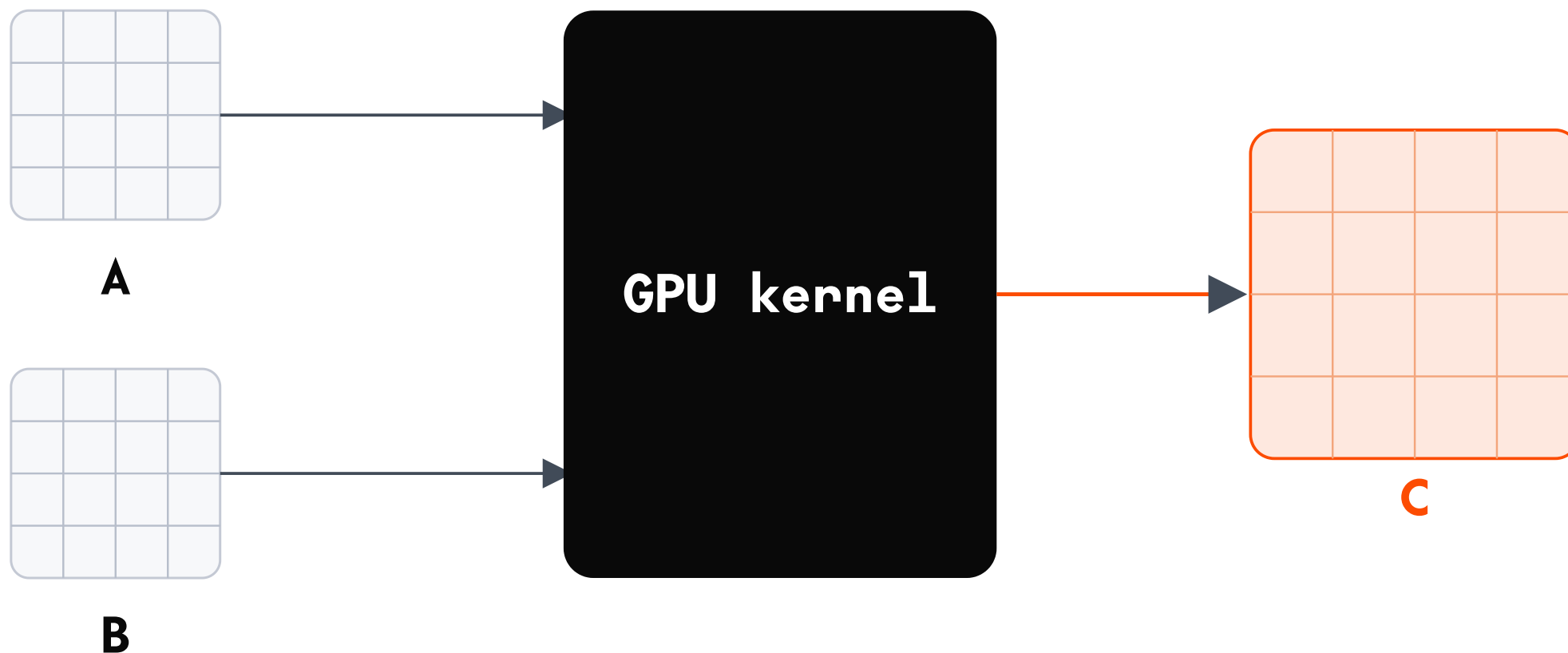


B

a math operation implemented for a specific chip

A kernel is a math operation **implemented** for a specific chip.

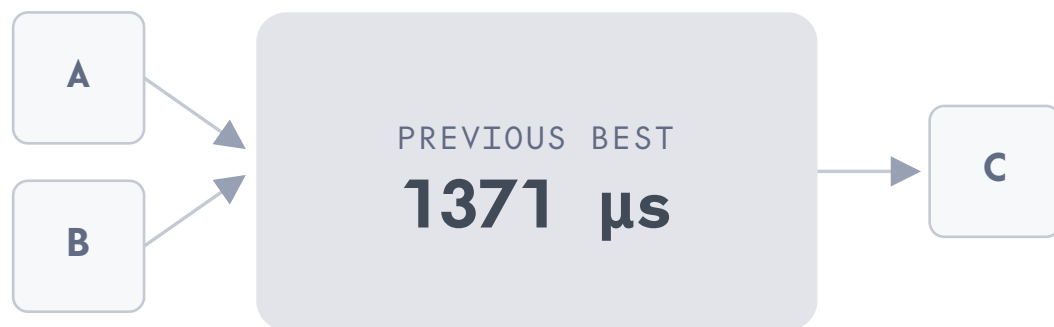
$$C = A \times B$$



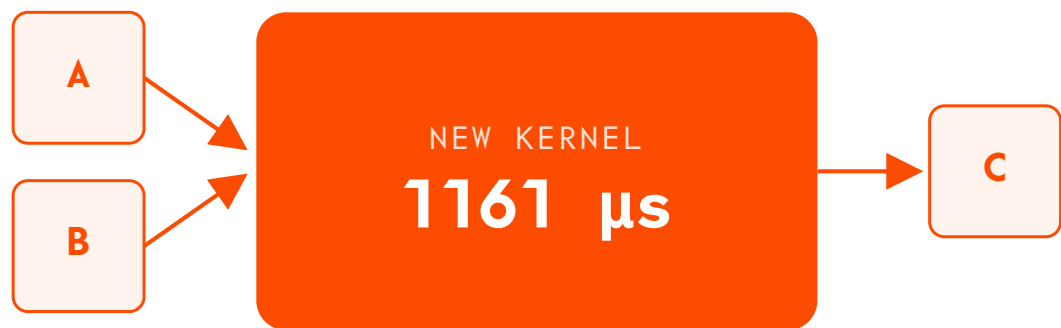
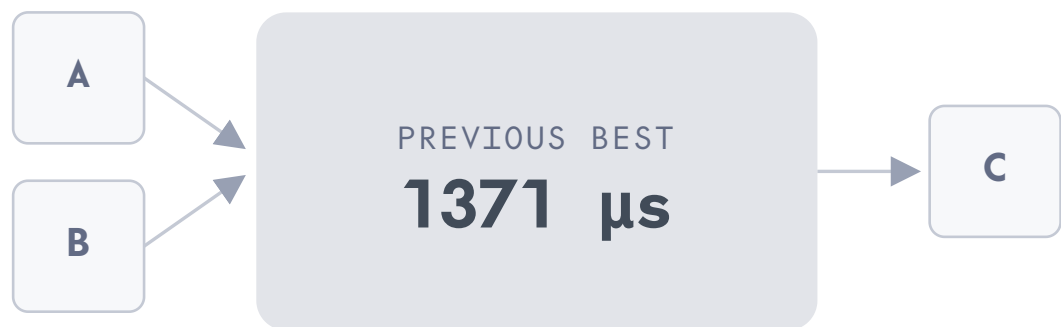
a math operation implemented for a specific chip

We want a kernel **faster** than what's already out there.

We want a kernel **faster** than what's already out there.



We want a kernel **faster** than what's already out there.



We want a kernel **faster** than what's already out there.



Today's kernel: TriMul

An operation in AlphaFold3

$$z_{ij} = \text{LayerNorm}\left(\sum_k a_{ik} \odot b_{jk}\right)$$

Swap TriMul for **any algorithm**: all you need is a verifier that measures the property you want to improve.

Humans competed on TriMul in the **GPUMode** kernel competition.

Open leaderboard run by **GPUMode** with NVIDIA and AMD.

HUMAN LEADERBOARD · TRIMUL · H100		runtime μ s
1st	Best human	1371
2nd	Human	2368
3rd	Human	2546
4th	Human	3655
5th	Human	4233

Everyone asks
Large
Language
Models now.

Write a Triton kernel for TriMul.

Schedule these jobs to minimize lateness.

Price this option with Monte Carlo.

Sort this in $O(n \log n)$, stable.

Assign shifts so nobody works two nights.

Rebalance the portfolio under a risk cap.

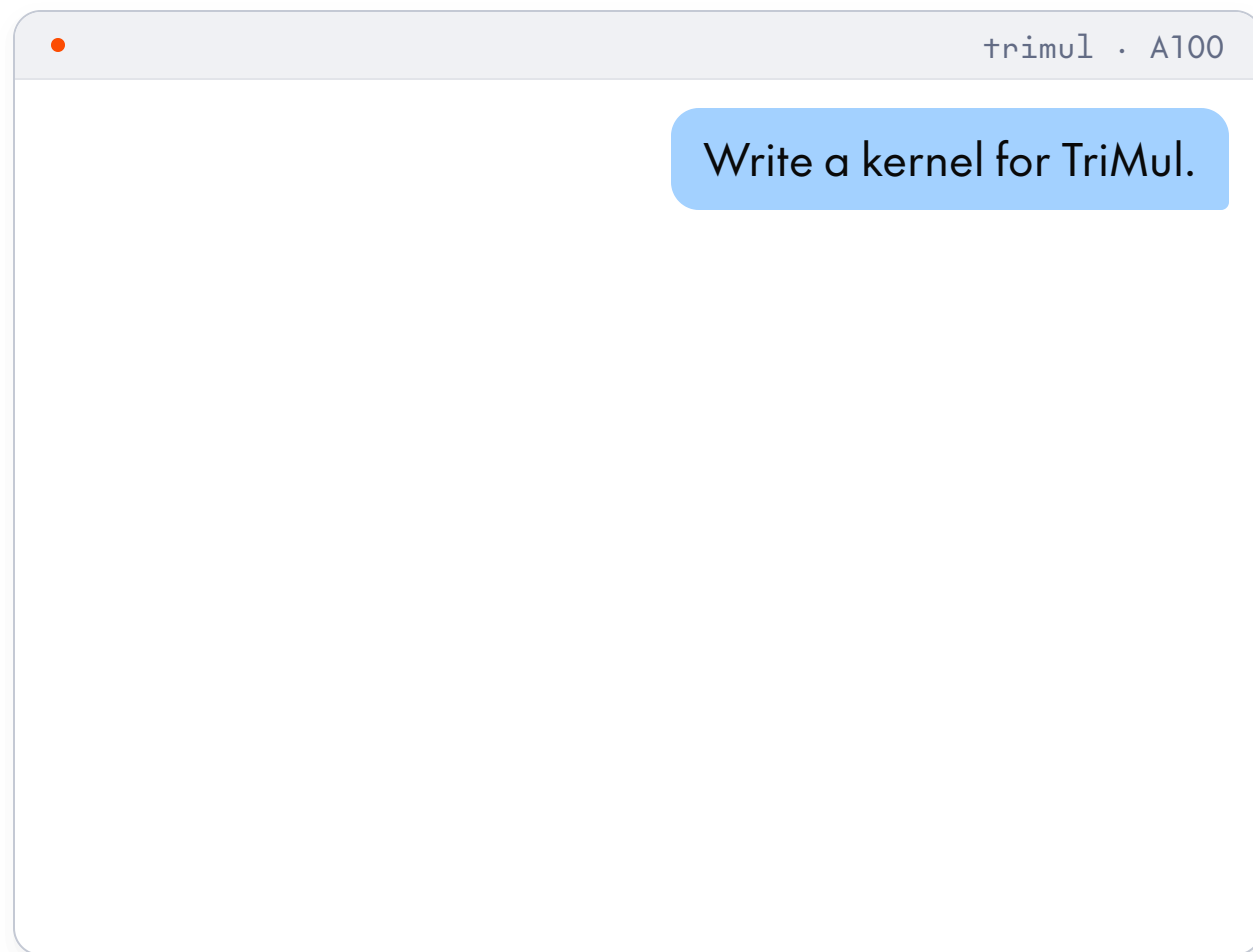
Implement Dijkstra with a heap.

Pack these tasks onto 8 GPUs.

Detect anomalies in this transaction log.

Turn this recursion into a DP.

Discovering new kernels **by hand**.



Discovering new kernels **by hand**.

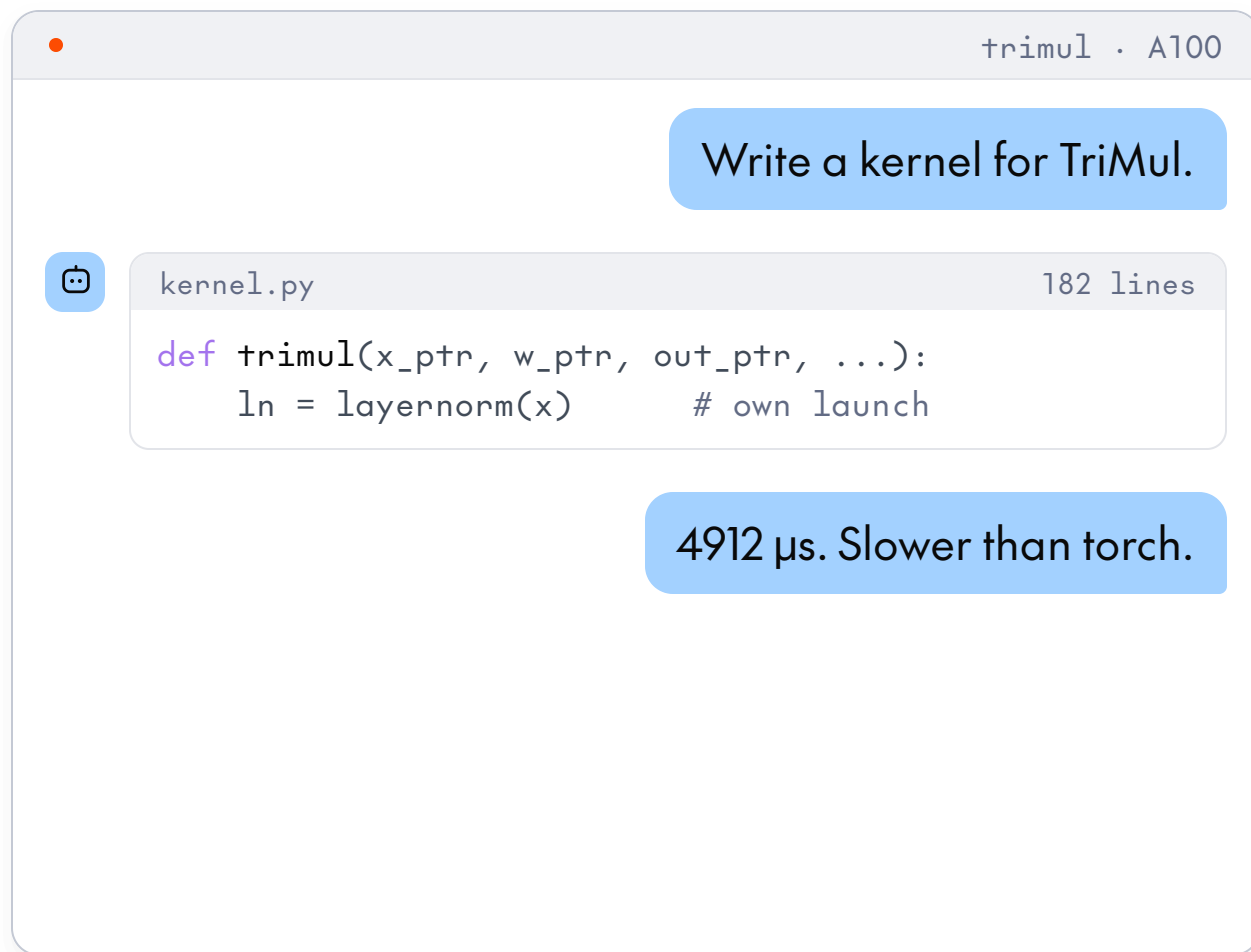
trimul . A100

Write a kernel for TriMul.

kernel.py 182 lines

```
def trimul(x_ptr, w_ptr, out_ptr, ...):  
    ln = layernorm(x)      # own launch
```

Discovering new kernels **by hand**.



Discovering new kernels **by hand**.

trimul · A100

Write a kernel for TriMul.

 kernel.py 182 lines

```
def trimul(x_ptr, w_ptr, out_ptr, ...):  
    ln = layernorm(x)      # own launch
```

4912 μ s. Slower than torch.

 kernel.py 206 lines

```
ln = fused_ln_proj(x)  # fp32 accum
```

Discovering new kernels **by hand**.

trimul · A100

Write a kernel for TriMul.

 kernel.py 182 lines

```
def trimul(x_ptr, w_ptr, out_ptr, ...):  
    ln = layernorm(x)    # own launch
```

4912 μ s. Slower than torch.

 kernel.py 206 lines

```
ln = fused_ln_proj(x) # fp32 accum
```

2640 μ s. The mask path still branches.

Discovering new kernels **by hand**.

trimul · A100

Write a kernel for TriMul.

kernel.py 182 lines

```
def trimul(x_ptr, w_ptr, out_ptr, ...):  
    ln = layernorm(x)      # own launch
```

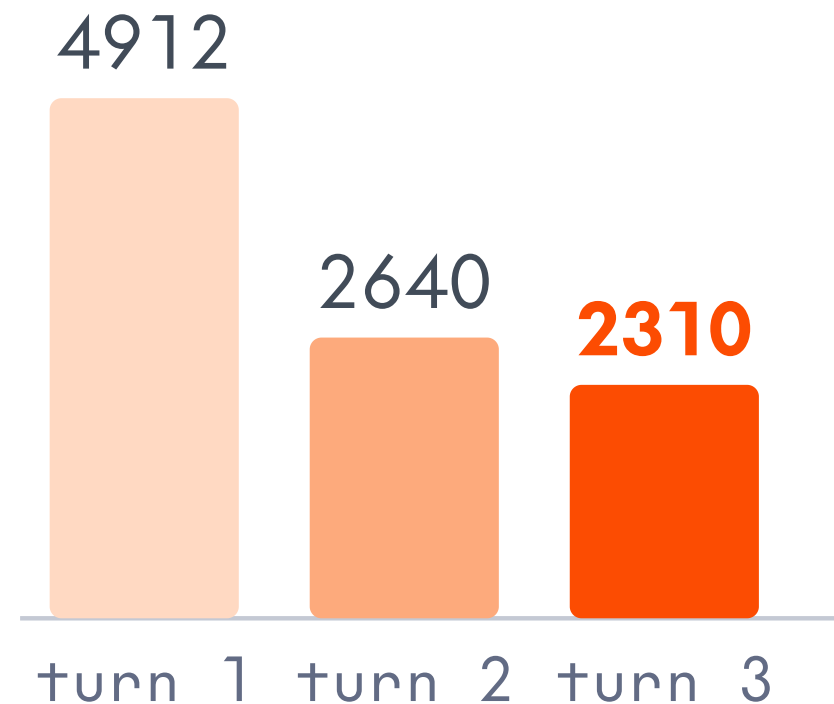
4912 μ s. Slower than torch.

kernel.py 206 lines

```
ln = fused_ln_proj(x)  # fp32 accum
```

2640 μ s. The mask path still branches.

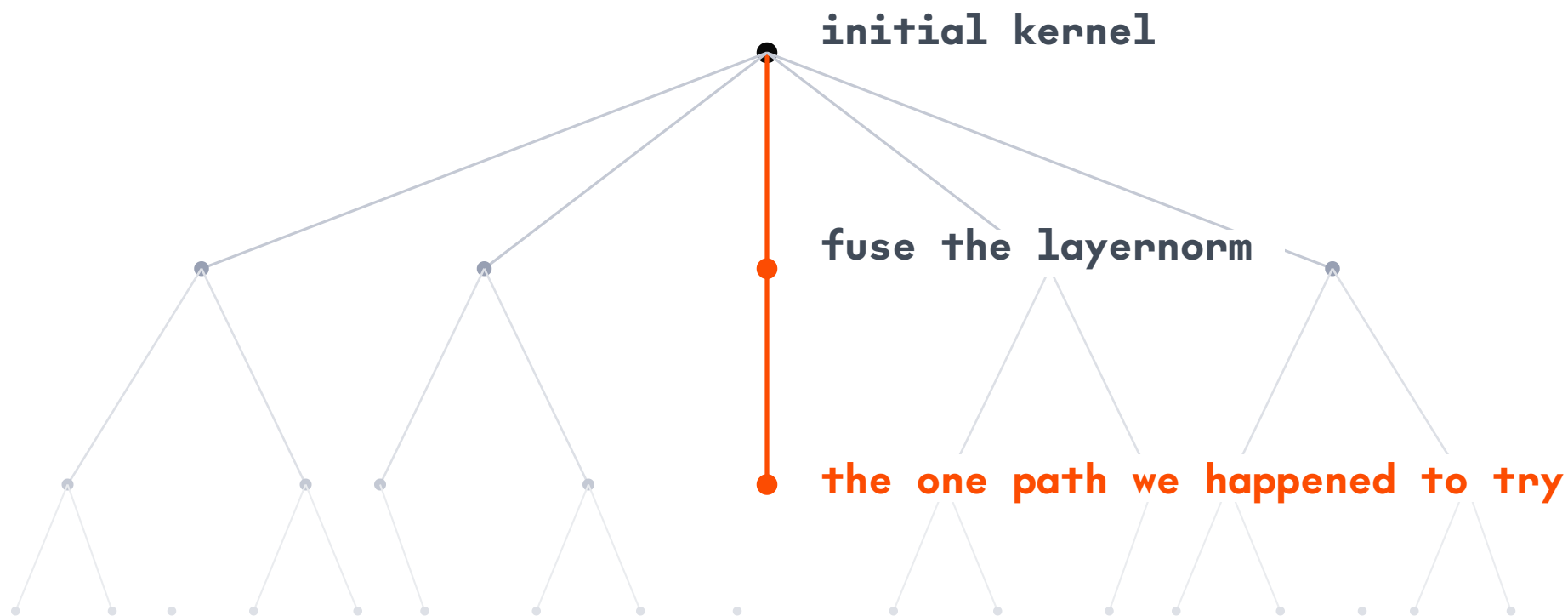
LATENCY PER TURN

 μ s · lower is better · illustrative

But there could have been **millions of other paths.**



But there could have been **millions of other paths.**



How do we **explore** this large space?

Some of these algorithms have **verifiers**.

Some of these algorithms have **verifiers**.

GPU KERNEL

```
def score(kernel): → -latency_us(kernel)
```

Some of these algorithms have **verifiers**.

GPU KERNEL

```
def score(kernel): → -latency_us(kernel)
```

SCHEDULING

```
def score(order): → contest_points(order)
```

Some of these algorithms have **verifiers**.

GPU KERNEL

```
def score(kernel): → -latency_us(kernel)
```

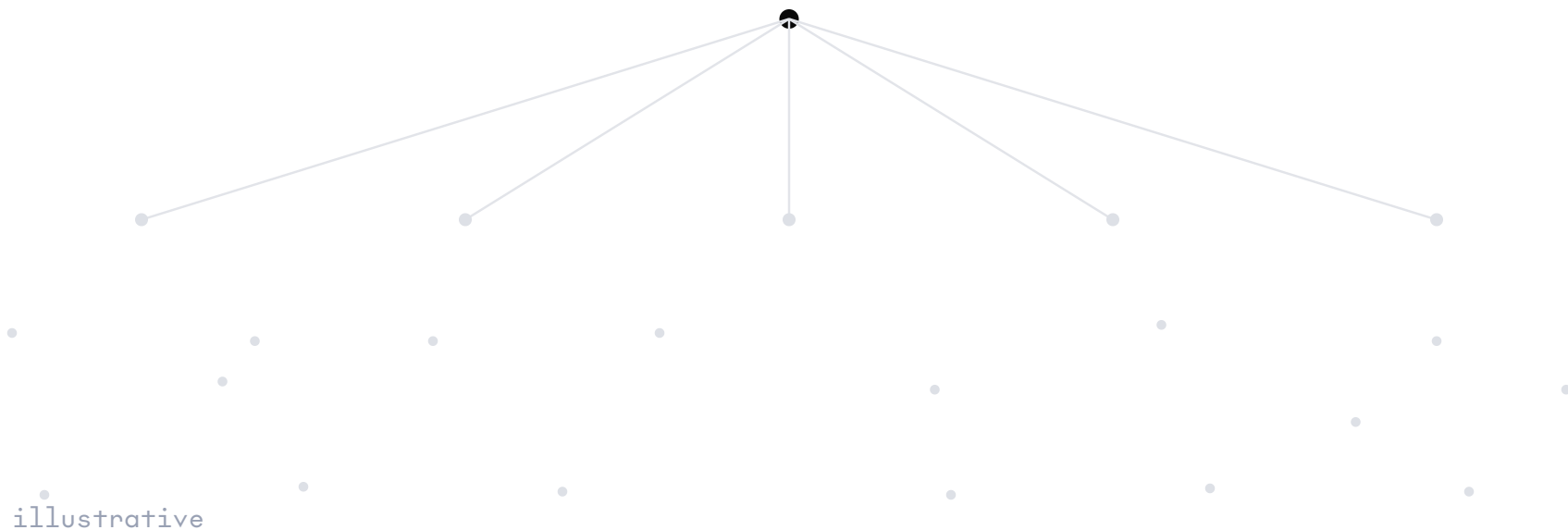
SCHEDULING

```
def score(order): → contest_points(order)
```

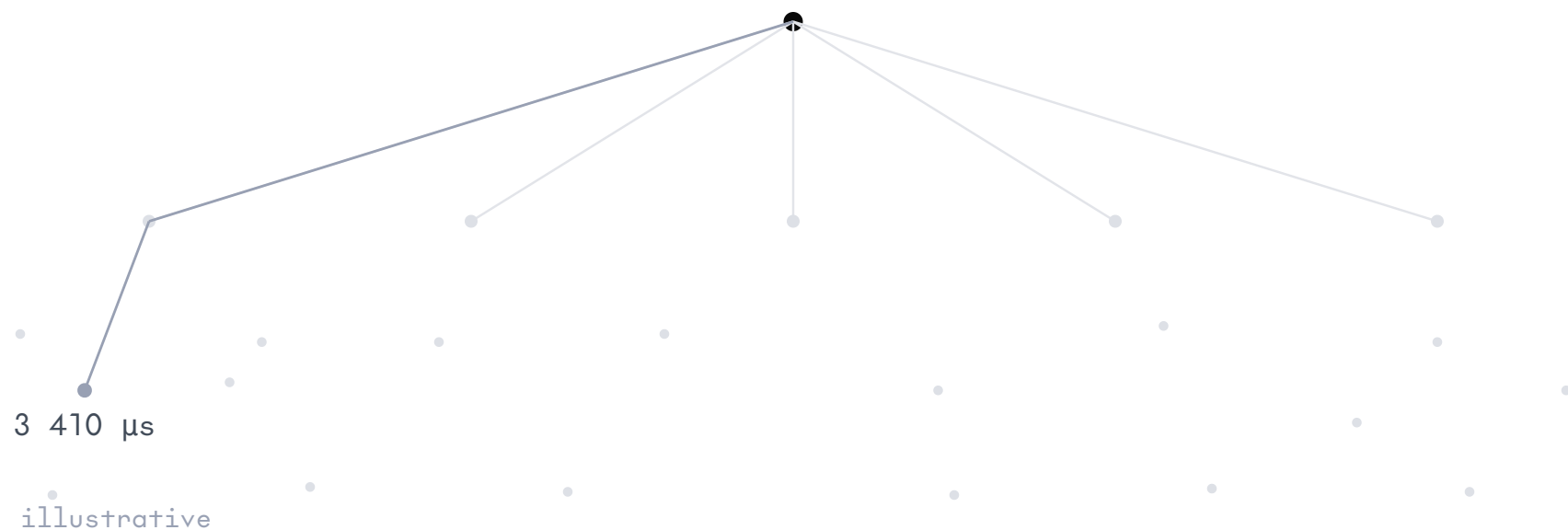
FINANCE

```
def score(w): → sharpe(w, returns)
```

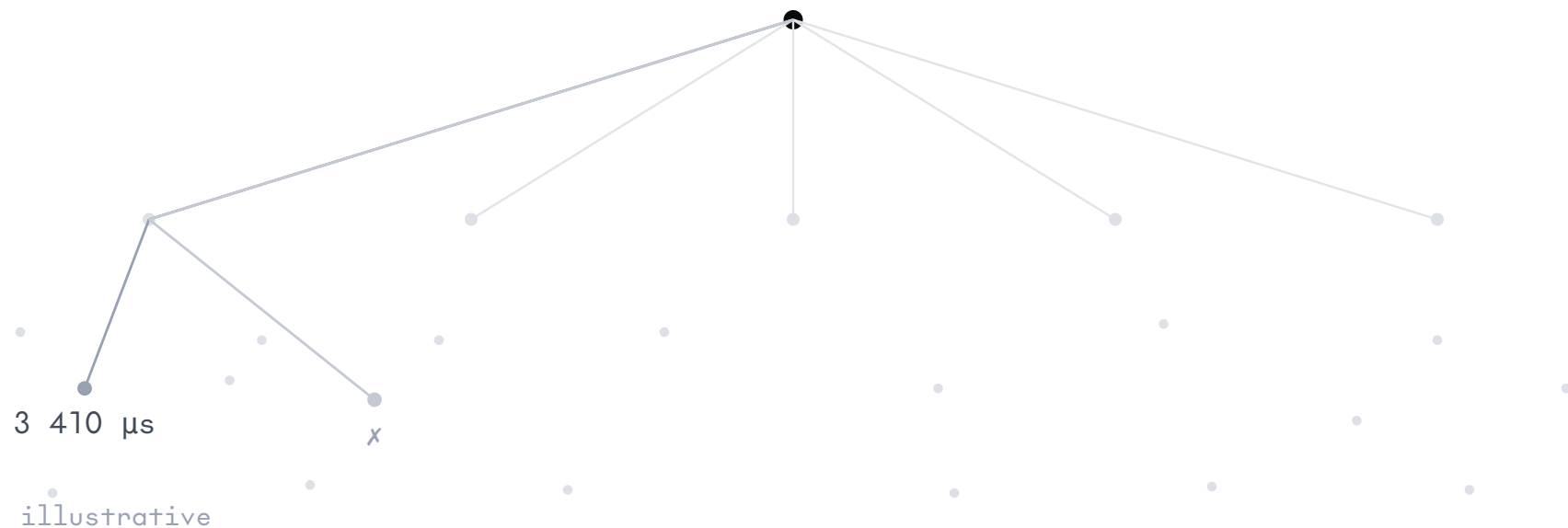
So we could just **sample at random** and verify.



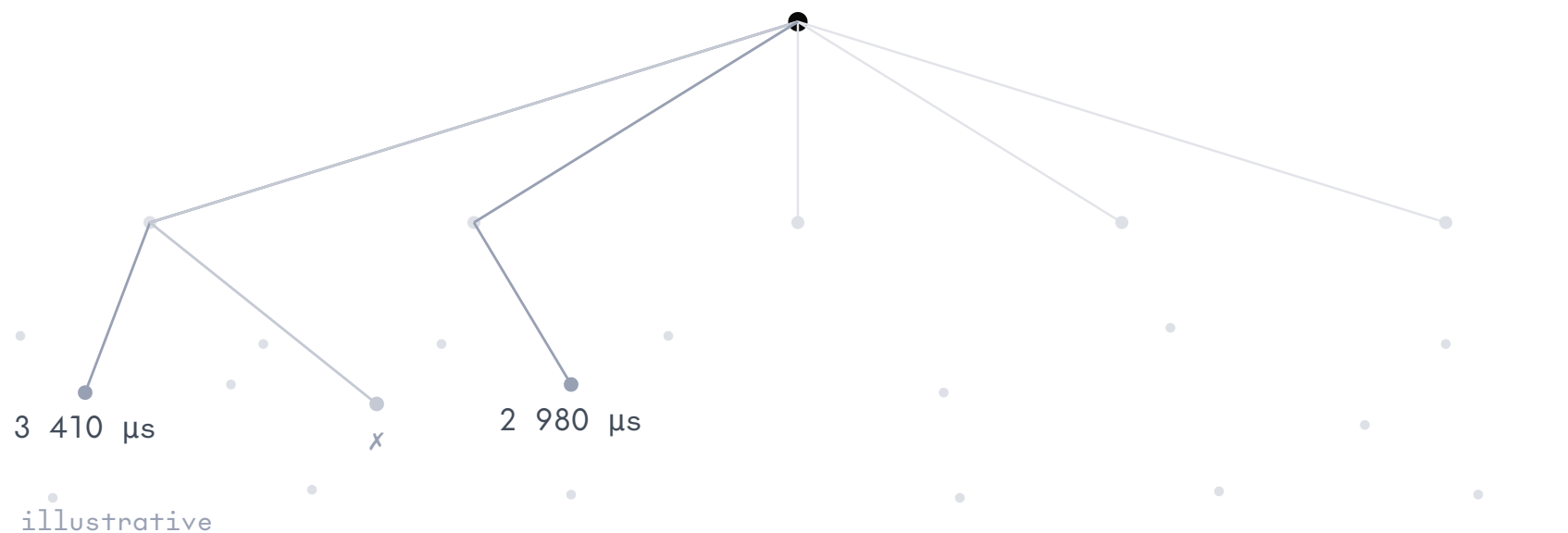
So we could just **sample at random** and verify.



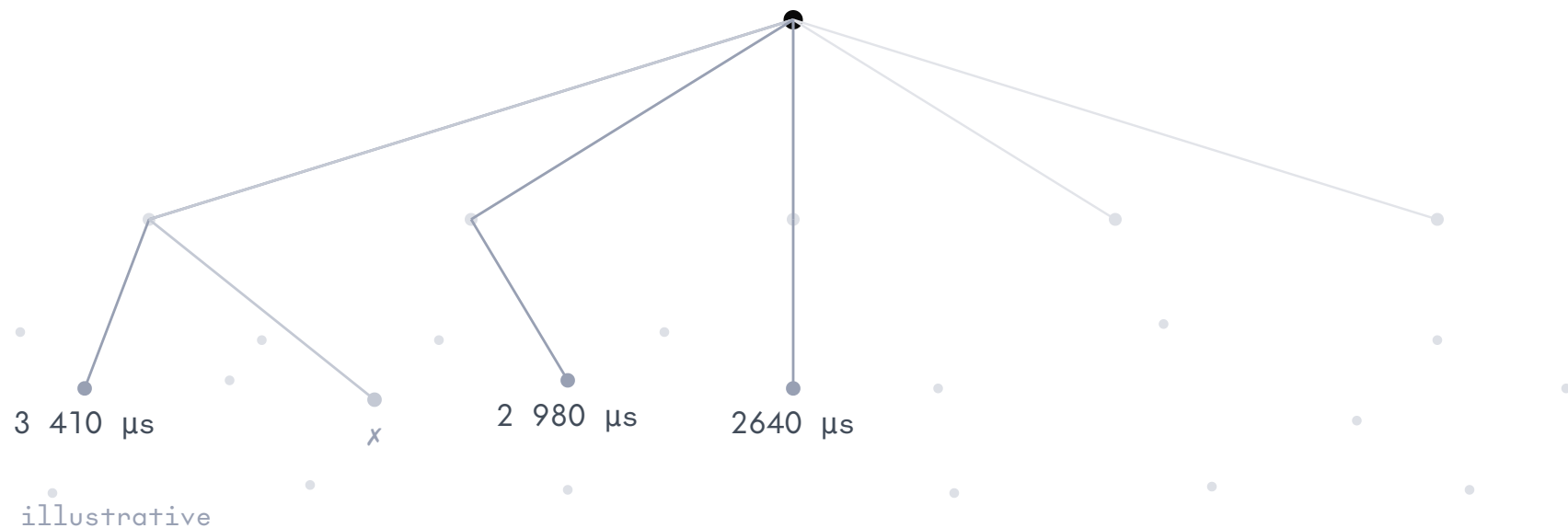
So we could just **sample at random** and verify.



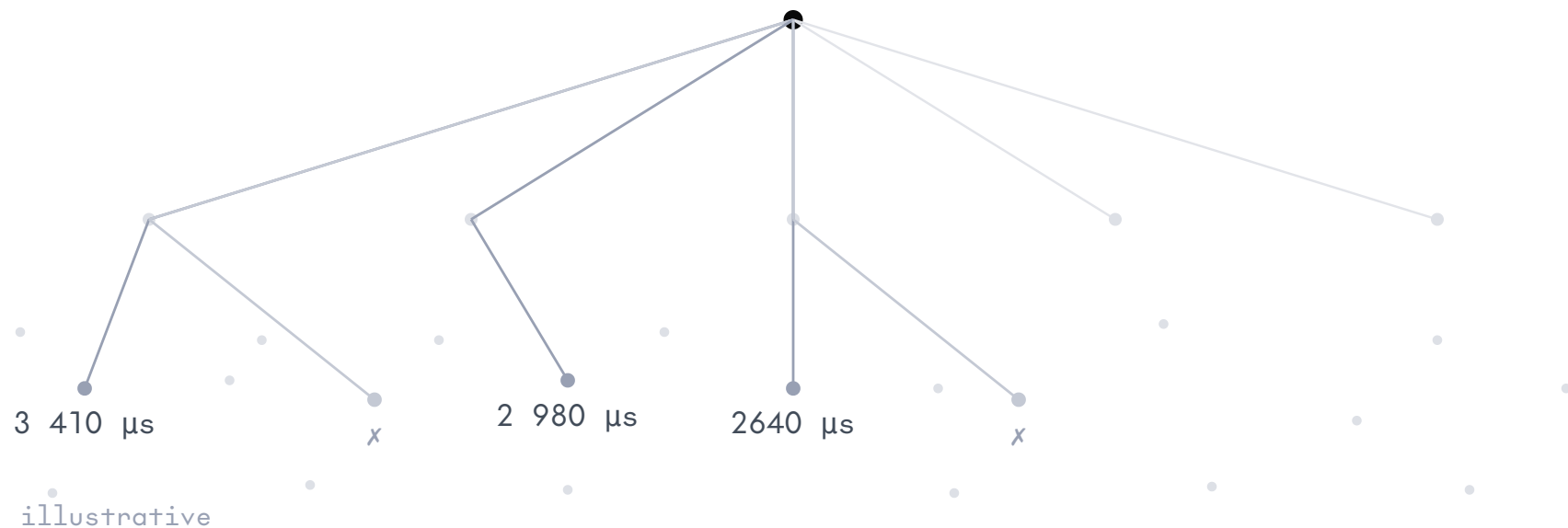
So we could just **sample at random** and verify.



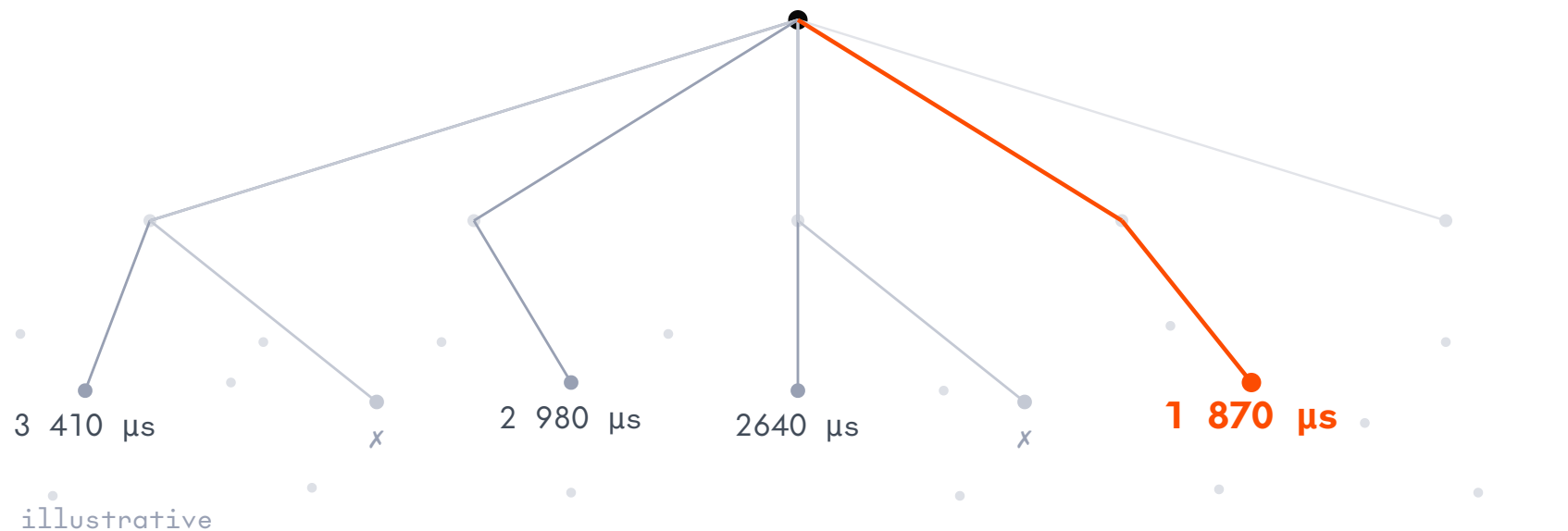
So we could just **sample at random** and verify.



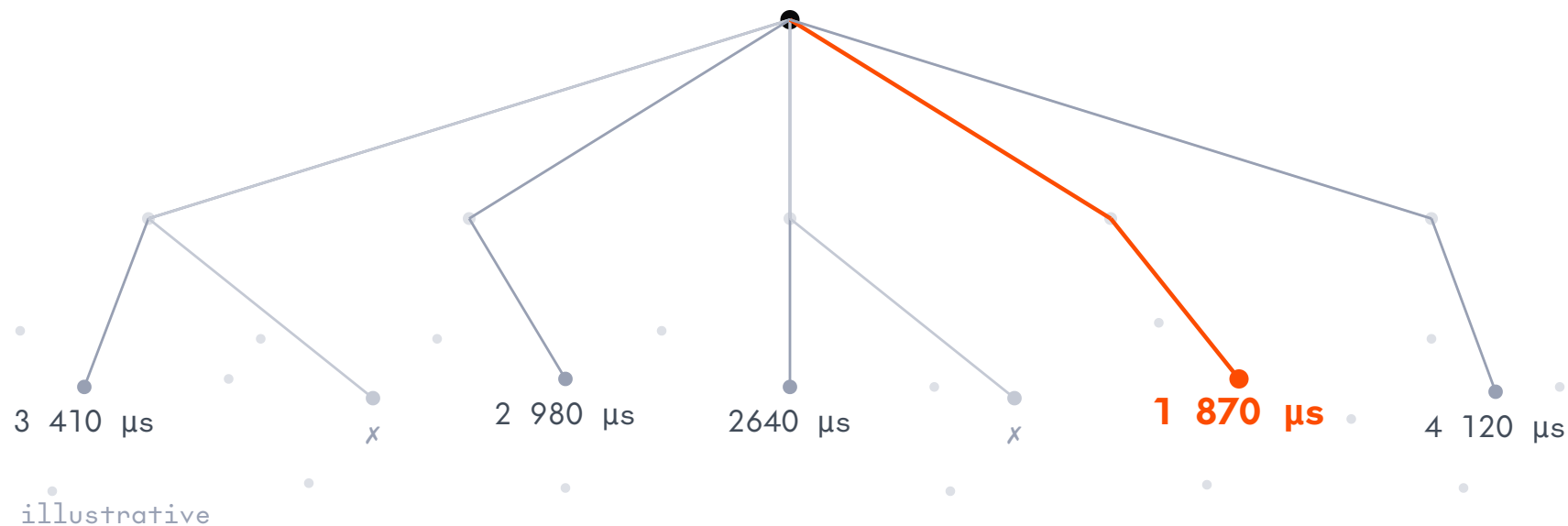
So we could just **sample at random** and verify.



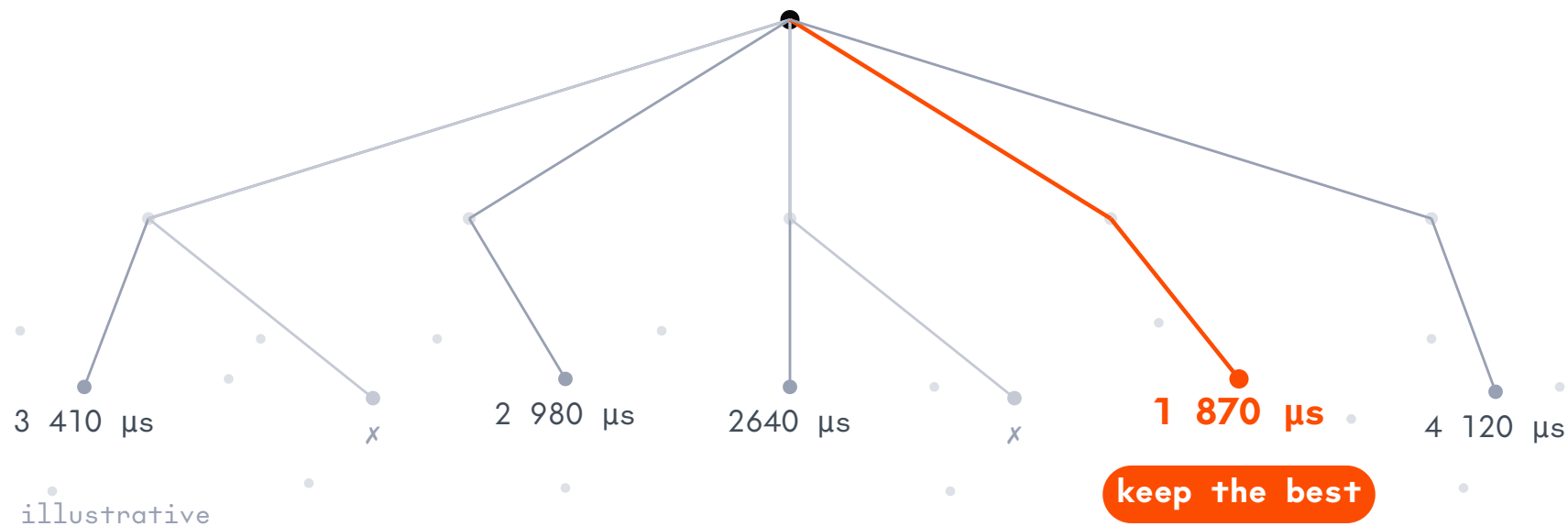
So we could just **sample at random** and verify.



So we could just **sample at random** and verify.



So we could just **sample at random** and verify.



But **random** is not really feasible.

But **random** is not really feasible.

random_tokens.py

```
def while( ( +l.import 7 => []  
elif offs ** return @ for(  
kernel[[ && lambda ;; x_ptr  
)))) if None class 0x3f  
+l.dot +l.dot +l.dot +l.dot  
...  
x 106 samples, 0 compile
```

- Random tokens will never compile.

But **random** is not really feasible.

random_tokens.py

```
def wh}le(( +l.import 7 => []  
elif offs ** return @ for(  
kernel[[ && lambda ;; x_ptr  
))))) if None class 0x3f  
+l.dot +l.dot +l.dot +l.dot  
...  
x 106 samples, 0 compile
```

- Random tokens will never compile.

grammar.py

```
for BLOCK in [32, 64, 128]:  
    for stages in [2, 3, 4]:  
        for warps in [4, 8]:  
            for layout in [row, col]:  
                kernel = template(BLOCK,  
                                   stages, warps, layout)  
✓ 36 programs, all valid
```

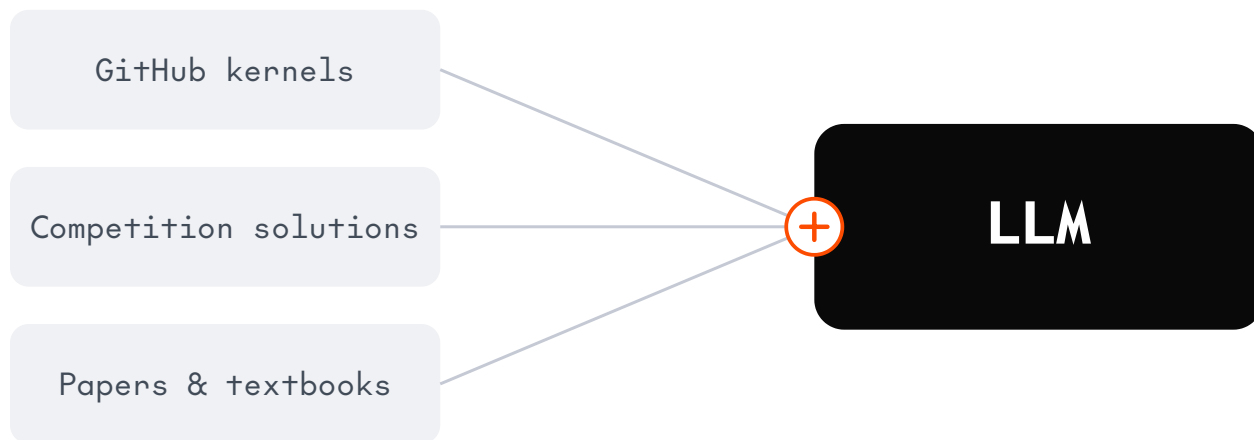
- A grammar limits our expressiveness.

LLMs are pretty good priors over the search space.

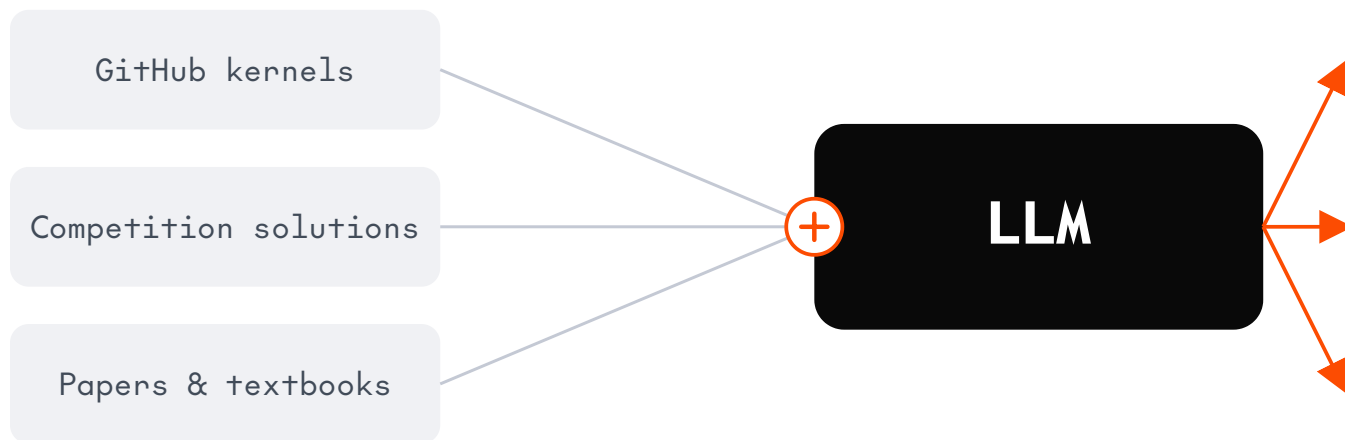
They've seen millions of solutions and can **recombine** them.



They've seen millions of solutions and can **recombine** them.



They've seen millions of solutions and can **recombine** them.



They've seen millions of solutions and can **recombine** them.



They've seen millions of solutions and can **recombine** them.

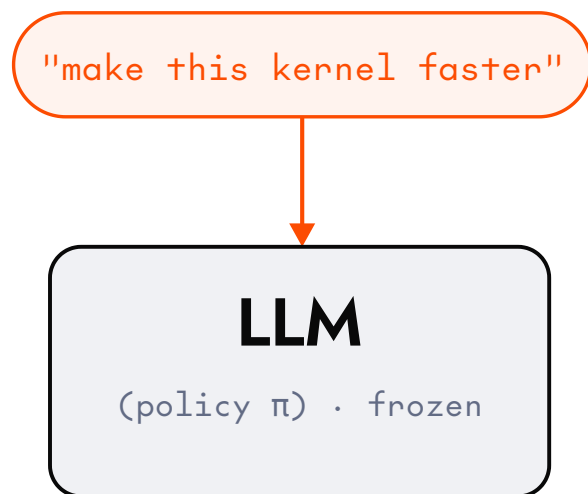


They've seen millions of solutions and can **recombine** them.

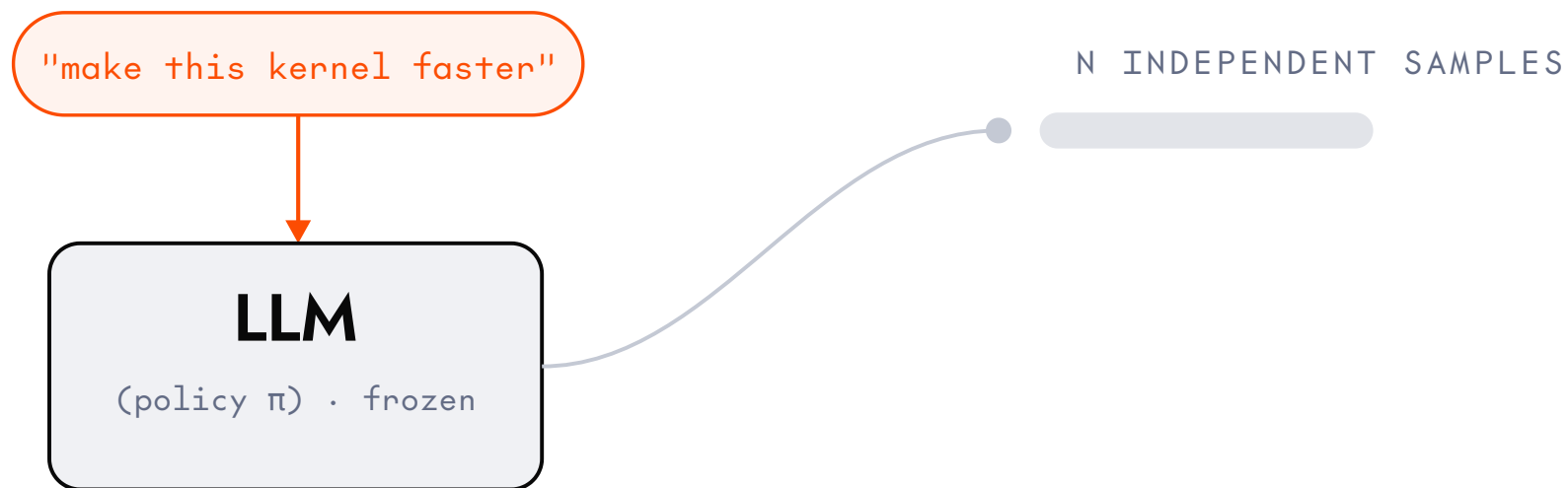


Let's sample many times from the LLM.

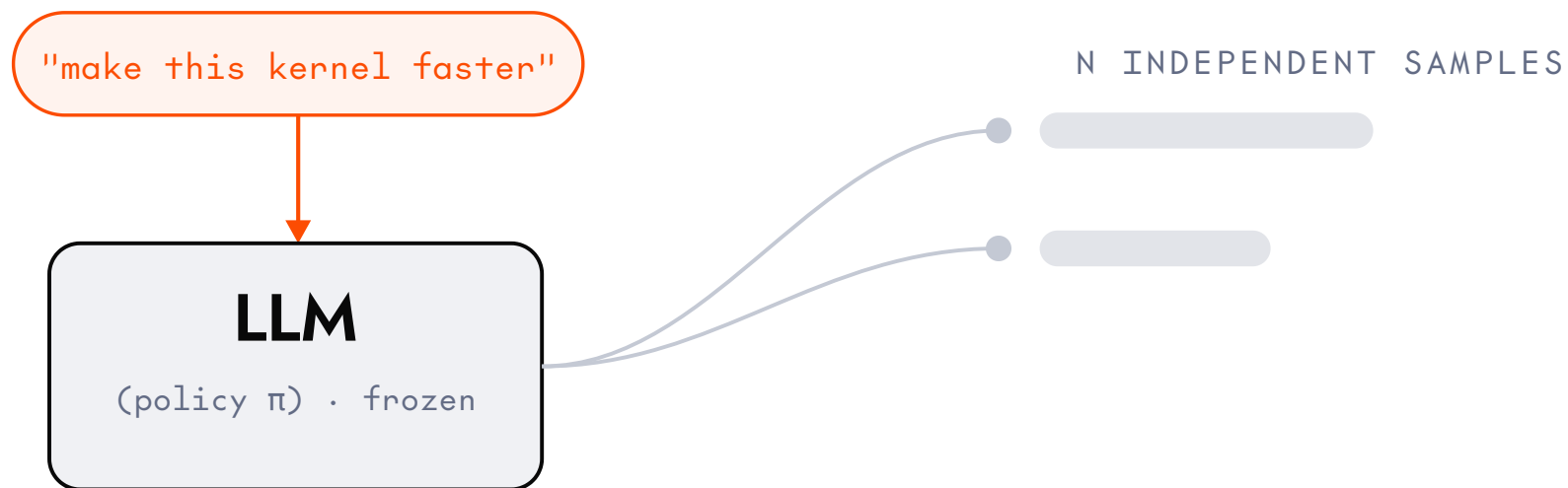
Let's **sample many times** from the LLM.



Let's sample many times from the LLM.



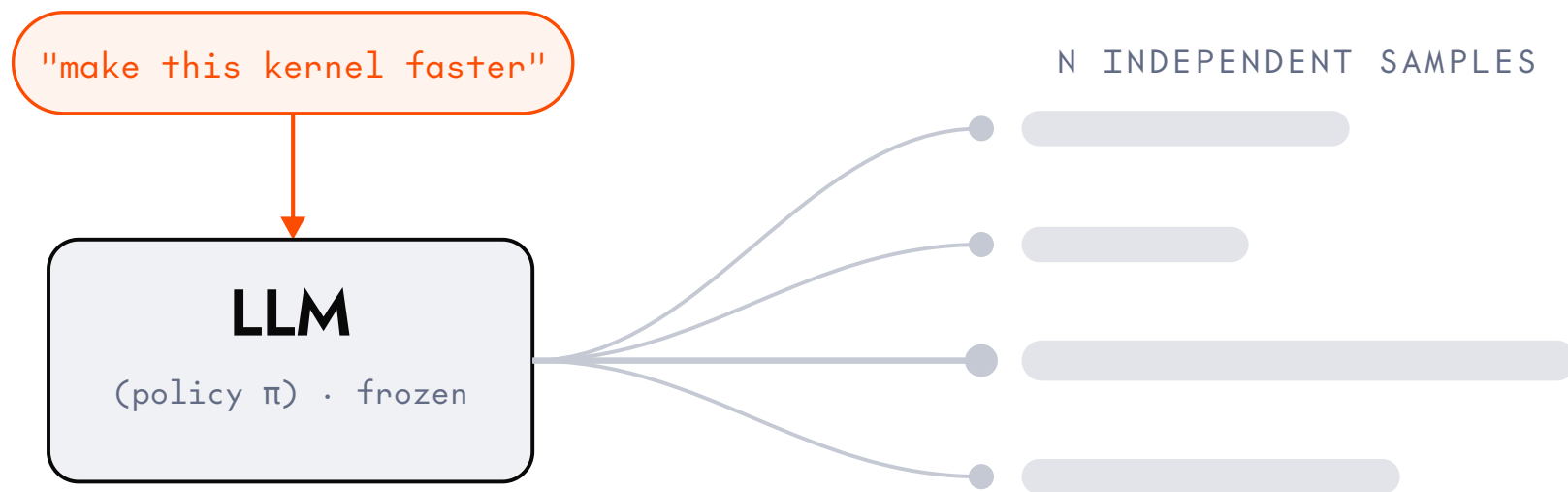
Let's **sample many times** from the LLM.



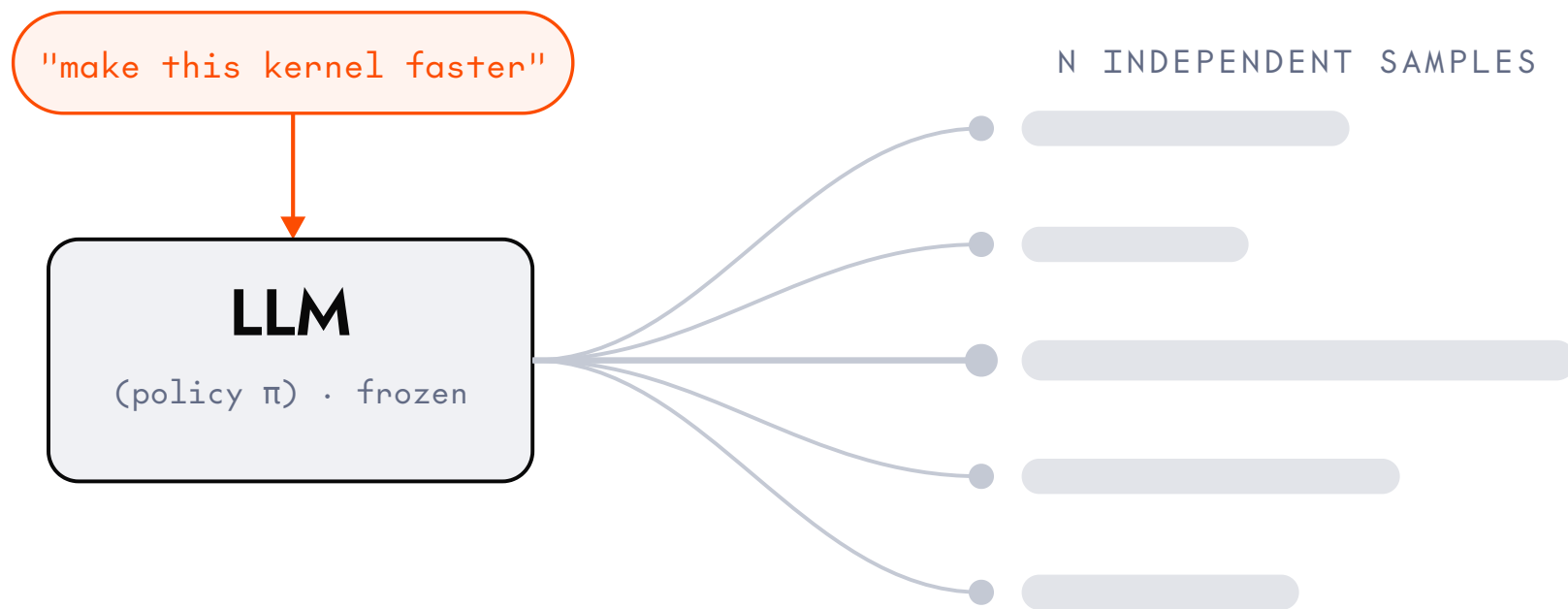
Let's **sample many times** from the LLM.



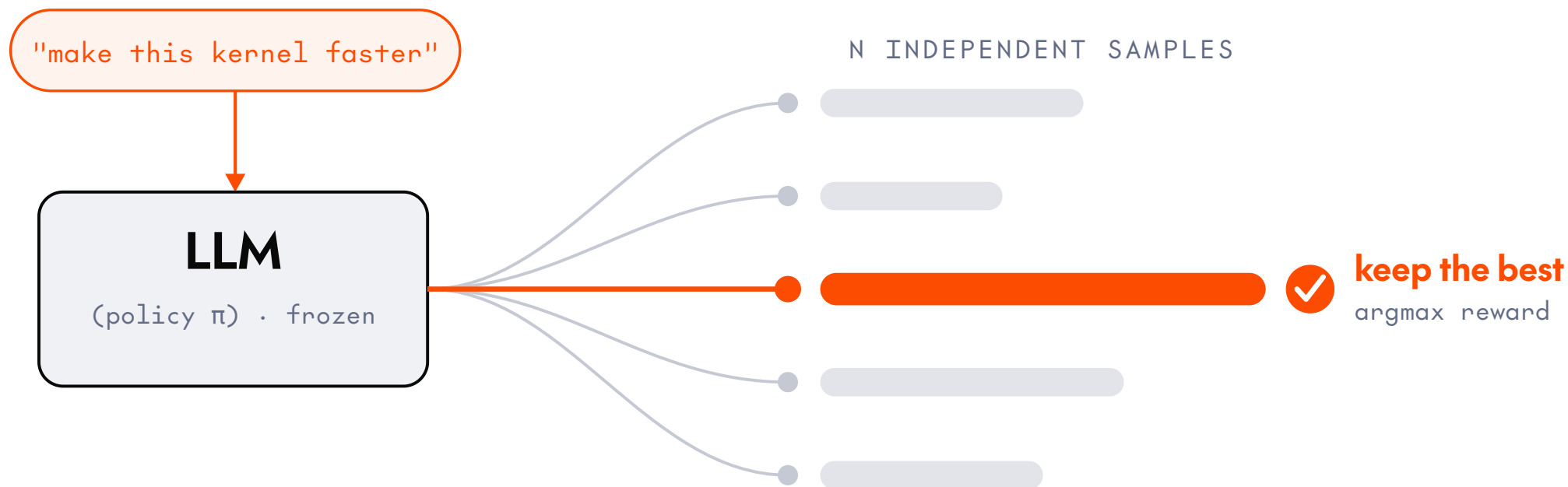
Let's **sample many times** from the LLM.



Let's **sample many times** from the LLM.

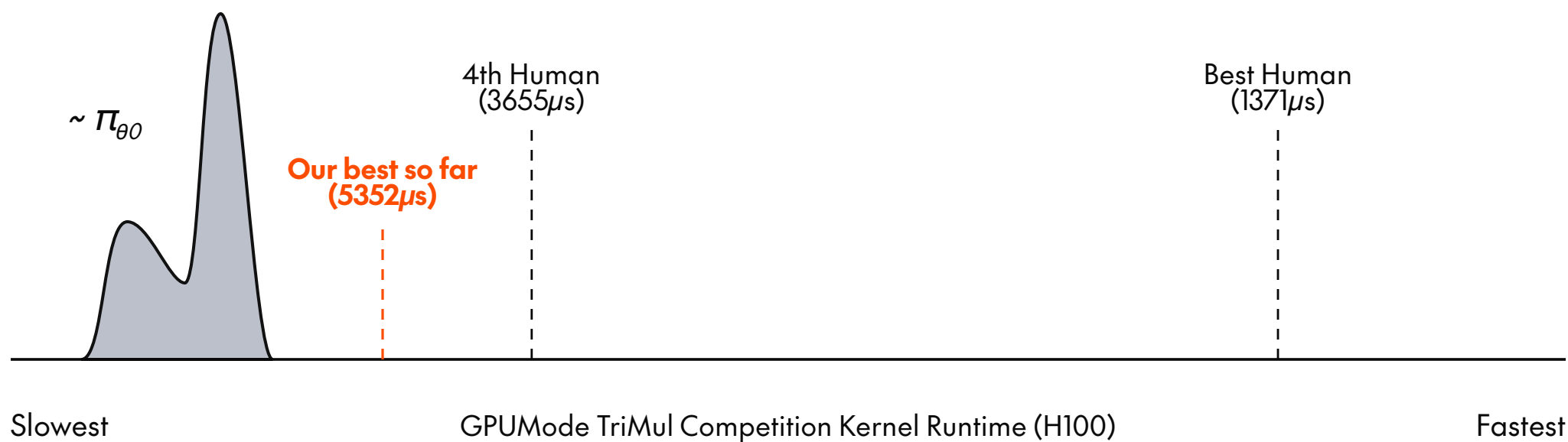


Let's **sample many times** from the LLM.



Let's take **gpt-oss-120b** from
OpenAI: how fast is a **TriMul**
kernel it samples?

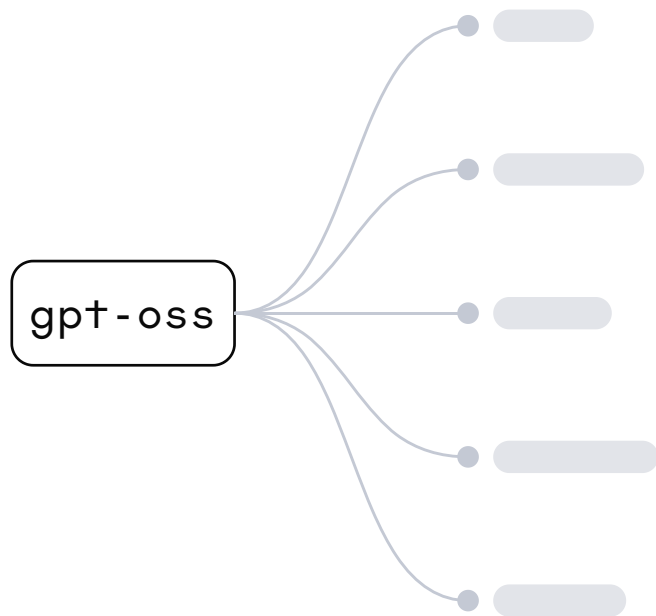
Sampling from the LLM is **not enough**.



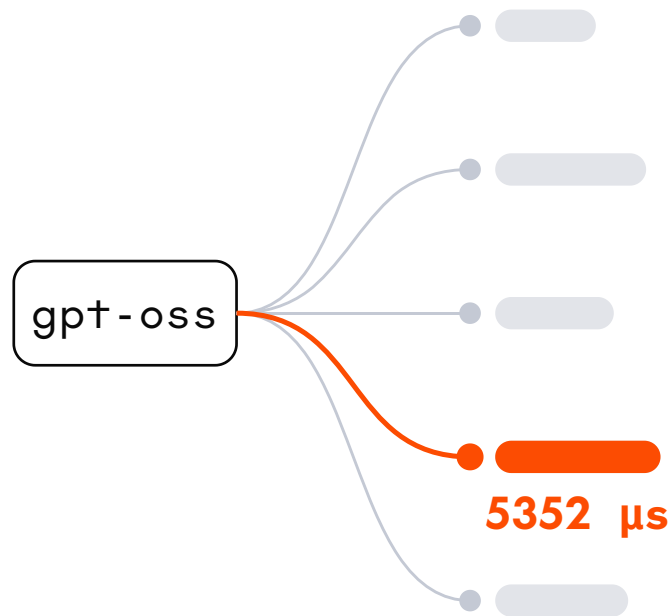
The problem: every sample starts from zero. We need to **build on what we found.**

Idea: Let's extend the horizon of the algorithm by **reusing states**

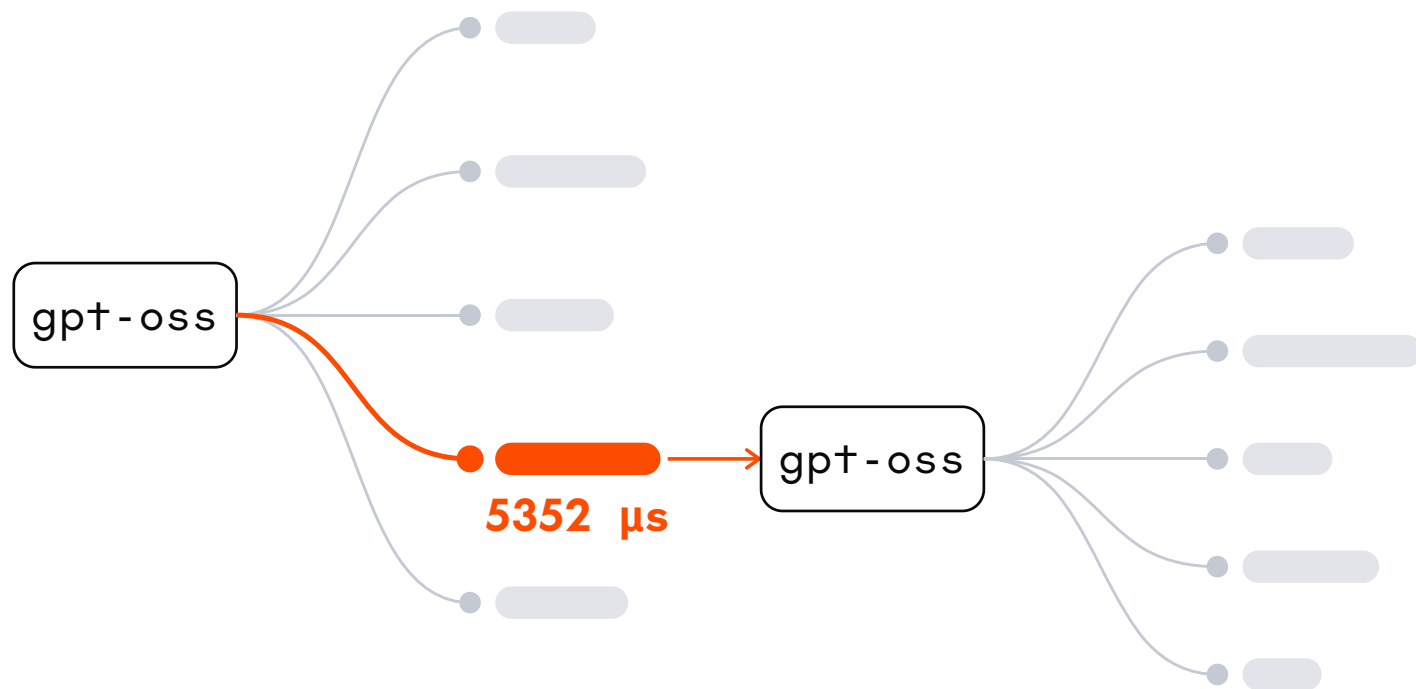
Idea: Let's extend the horizon of the algorithm by **reusing states**



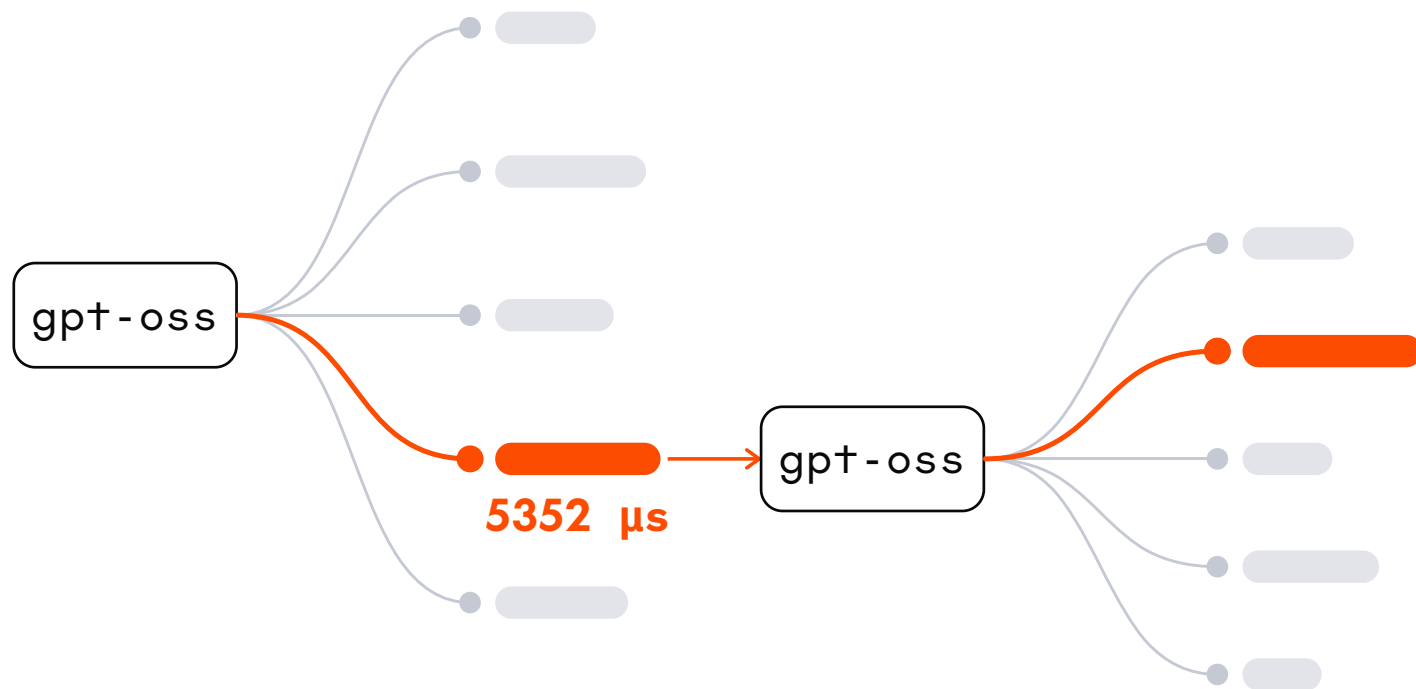
Idea: Let's extend the horizon of the algorithm by **reusing states**



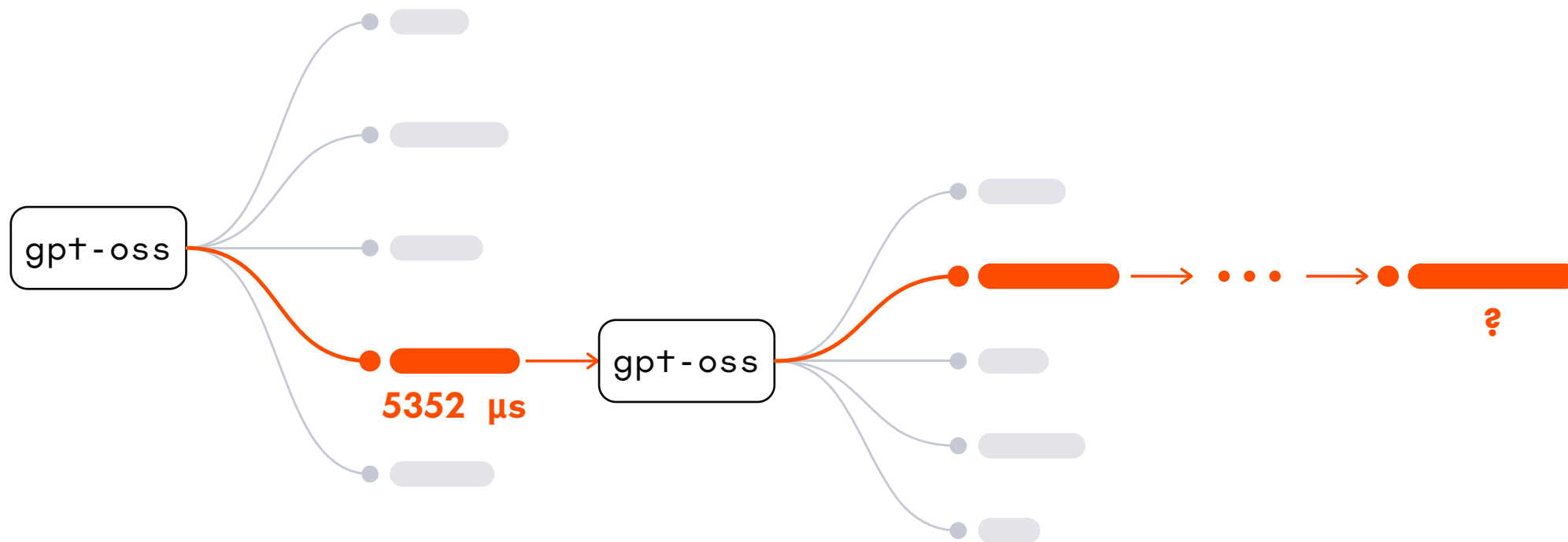
Idea: Let's extend the horizon of the algorithm by **reusing states**



Idea: Let's extend the horizon of the algorithm by **reusing states**



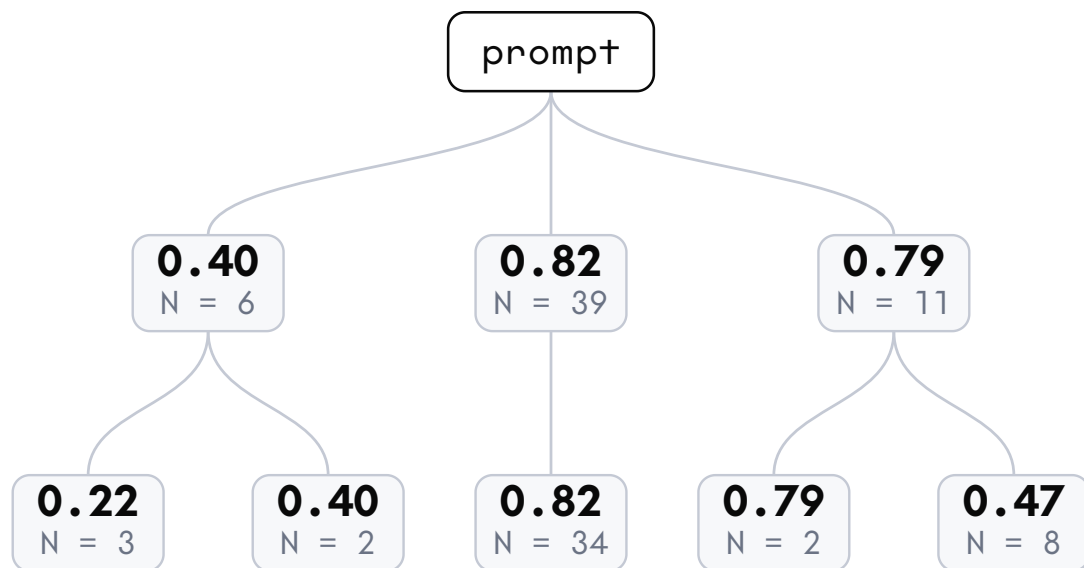
Idea: Let's extend the horizon of the algorithm by **reusing states**



But always resuming the **best** digs a hole.

Pure exploitation: we **never see the other branches**.

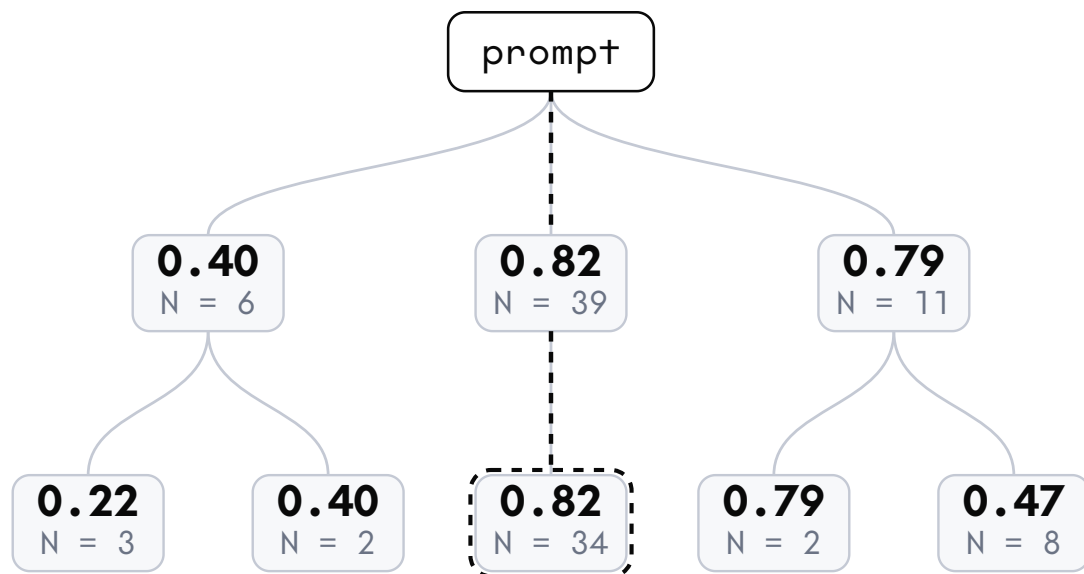
Monte Carlo Tree Search inspired approach to **balance exploration and exploitation**



$$\arg \max_s Q(s) + c P(s) \frac{\sqrt{1+T}}{1+n(s)}$$

- Q best score found below it: a state is worth its best child
- P rank of the state by reward
- N how often we resumed it or its descendants

Monte Carlo Tree Search inspired approach to **balance exploration and exploitation**

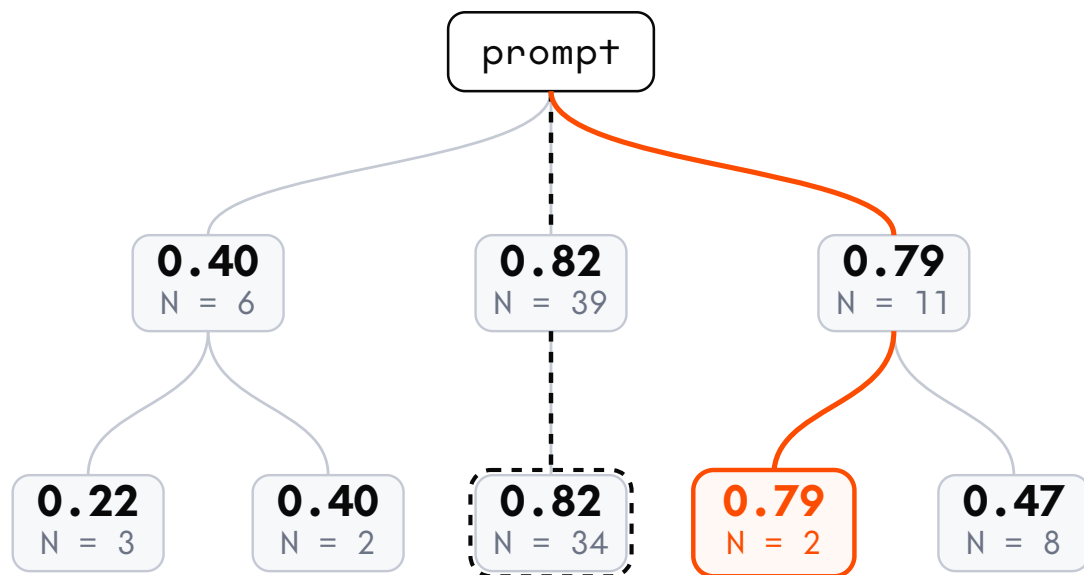


best-first: resume the top score, again

$$\arg \max_s Q(s) + c P(s) \frac{\sqrt{1+T}}{1+n(s)}$$

- Q best score found below it: a state is worth its best child
- P rank of the state by reward
- N how often we resumed it or its descendants

Monte Carlo Tree Search inspired approach to **balance exploration and exploitation**



best-first: resume the top score, again

PUCT: nearly as good, barely explored

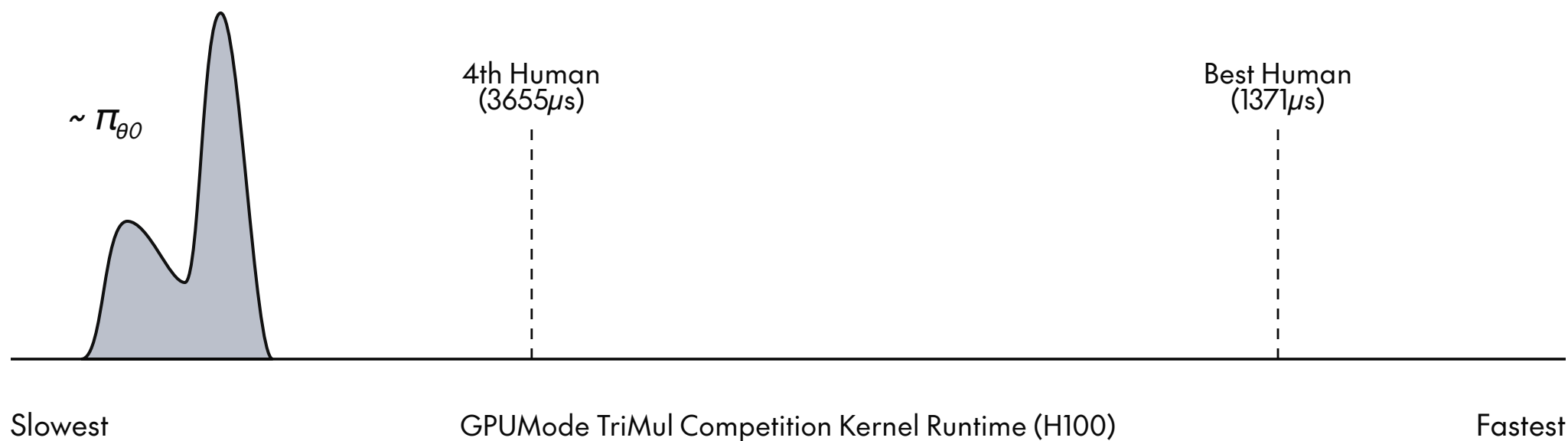
$$\arg \max_s Q(s) + c P(s) \frac{\sqrt{1+T}}{1+n(s)}$$

Q best score found below it: a state is worth its best child

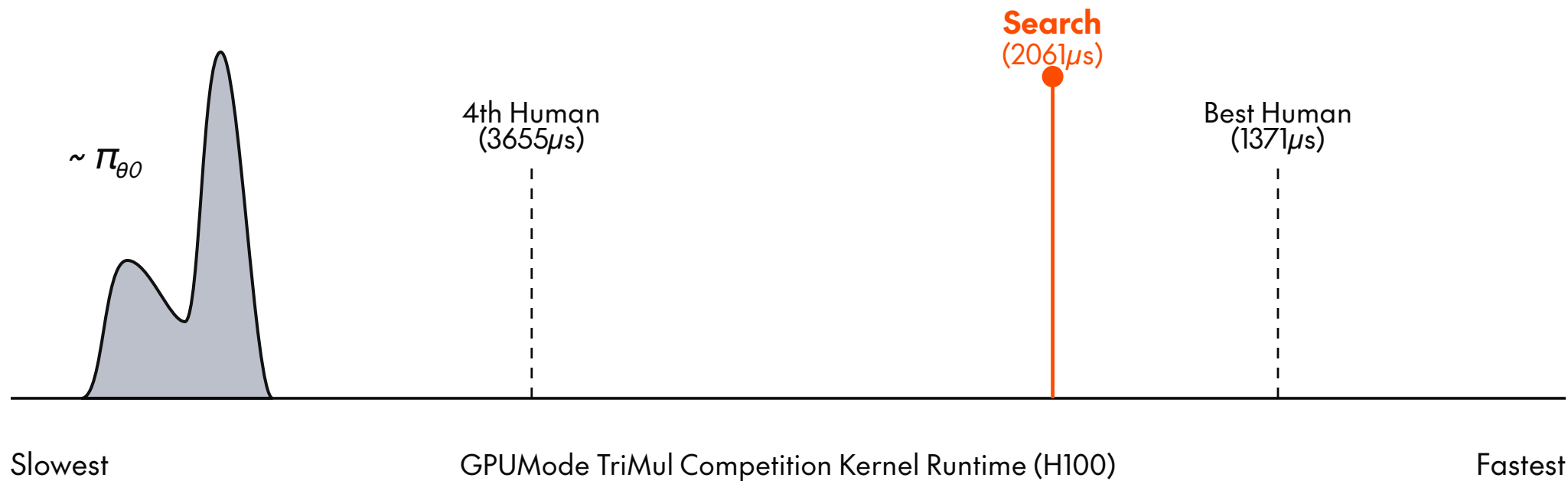
P rank of the state by reward

N how often we resumed it or its descendants

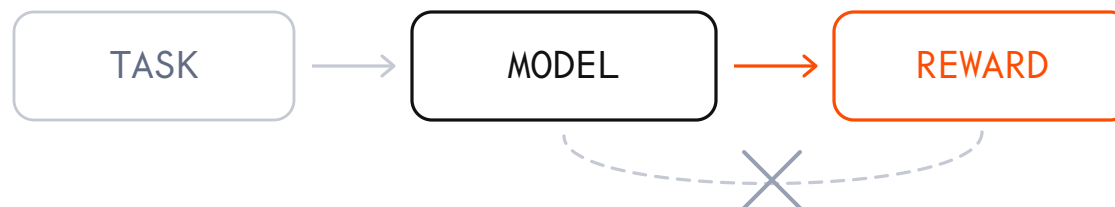
State reuse gives us a faster kernel.



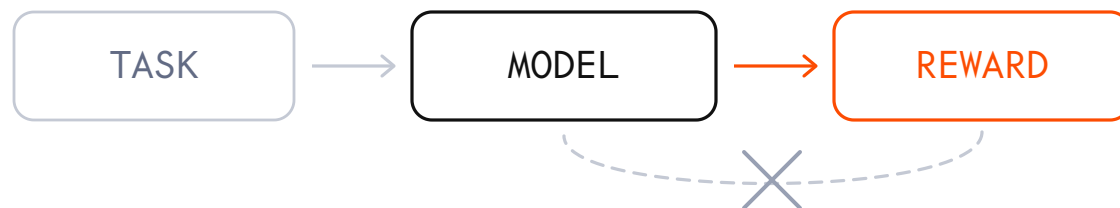
State reuse gives us a faster kernel.



Problem: the model never **learns**.



Problem: the model never **learns**.



Search got us from 5352 to 2061 μ s, but gpt-oss is exactly as good at writing kernels as when we started.

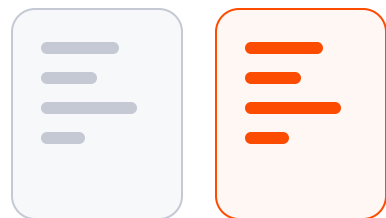
And the model starts out **bad** at writing kernels.

And the model starts out **bad** at writing kernels.



x

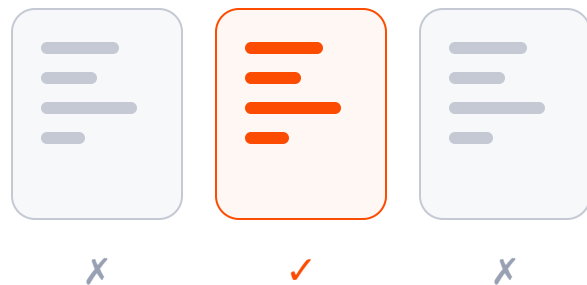
And the model starts out **bad** at writing kernels.



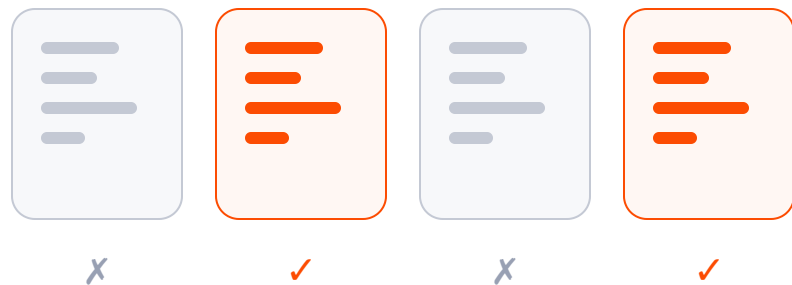
x

✓

And the model starts out **bad** at writing kernels.



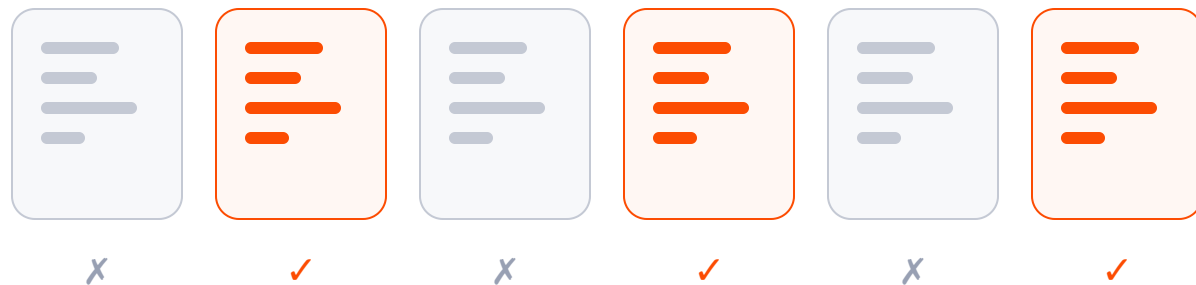
And the model starts out **bad** at writing kernels.



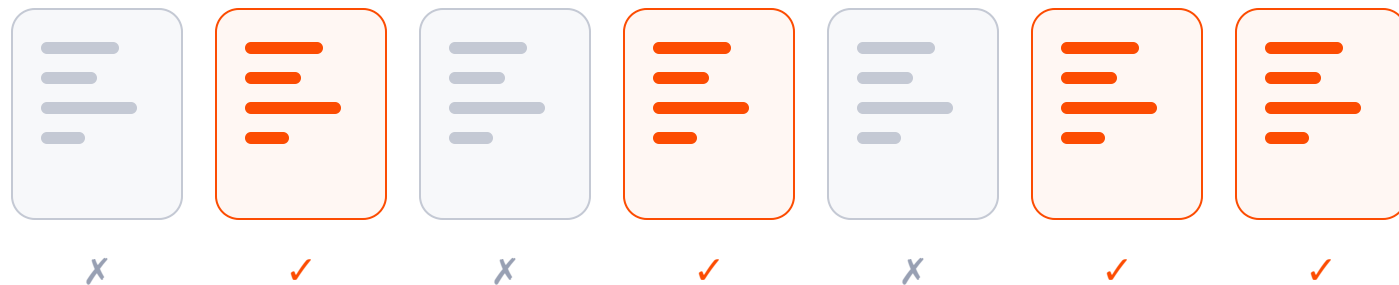
And the model starts out **bad** at writing kernels.



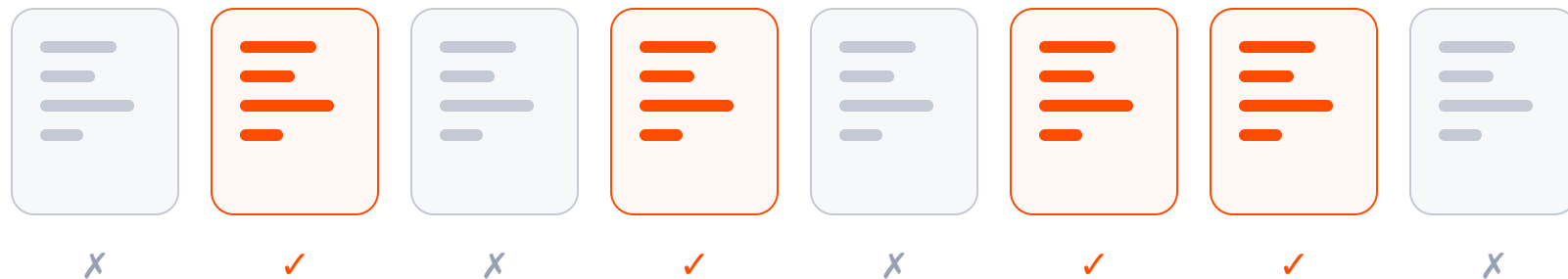
And the model starts out **bad** at writing kernels.



And the model starts out **bad** at writing kernels.



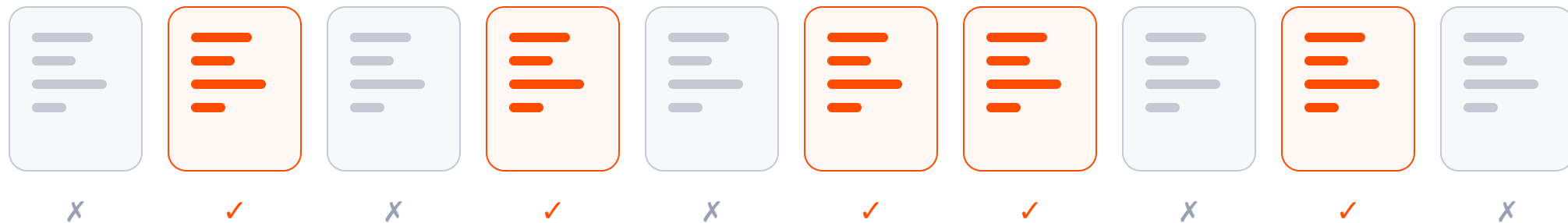
And the model starts out **bad** at writing kernels.



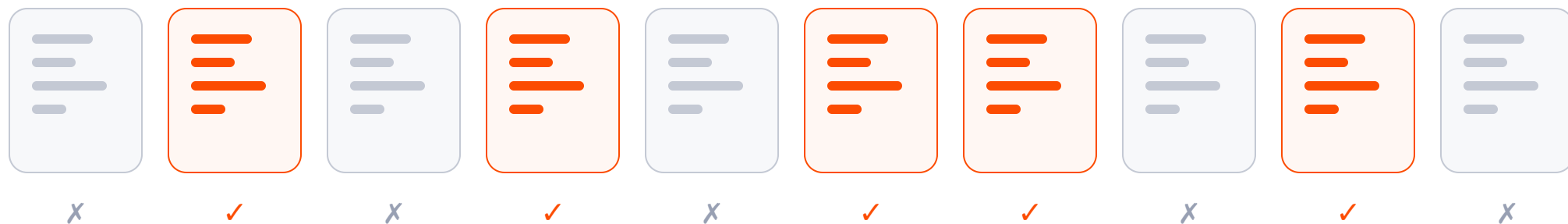
And the model starts out **bad** at writing kernels.



And the model starts out **bad** at writing kernels.



And the model starts out **bad** at writing kernels.



Only **46%** of the TriMul kernels gpt-oss writes pass the correctness check. And that is before counting the ones that are simply slow.

And it keeps trying the **same thing** over and over.

And it keeps trying the **same thing** over and over.

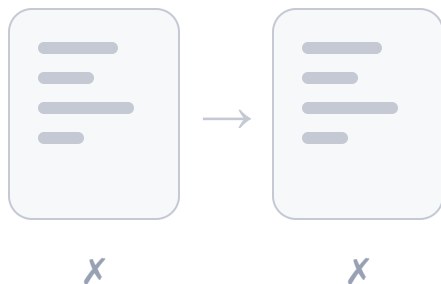


x

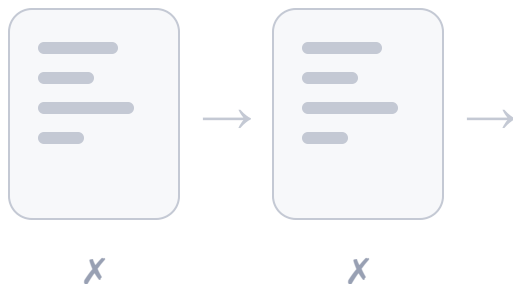
And it keeps trying the **same thing** over and over.



And it keeps trying the **same thing** over and over.



And it keeps trying the **same thing** over and over.



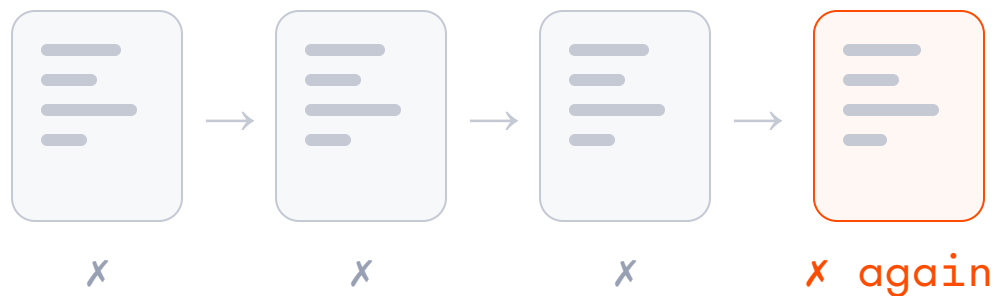
And it keeps trying the **same thing** over and over.



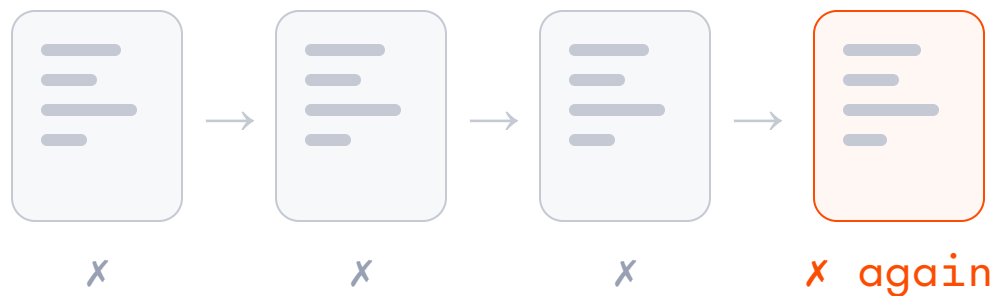
And it keeps trying the **same thing** over and over.



And it keeps trying the **same thing** over and over.

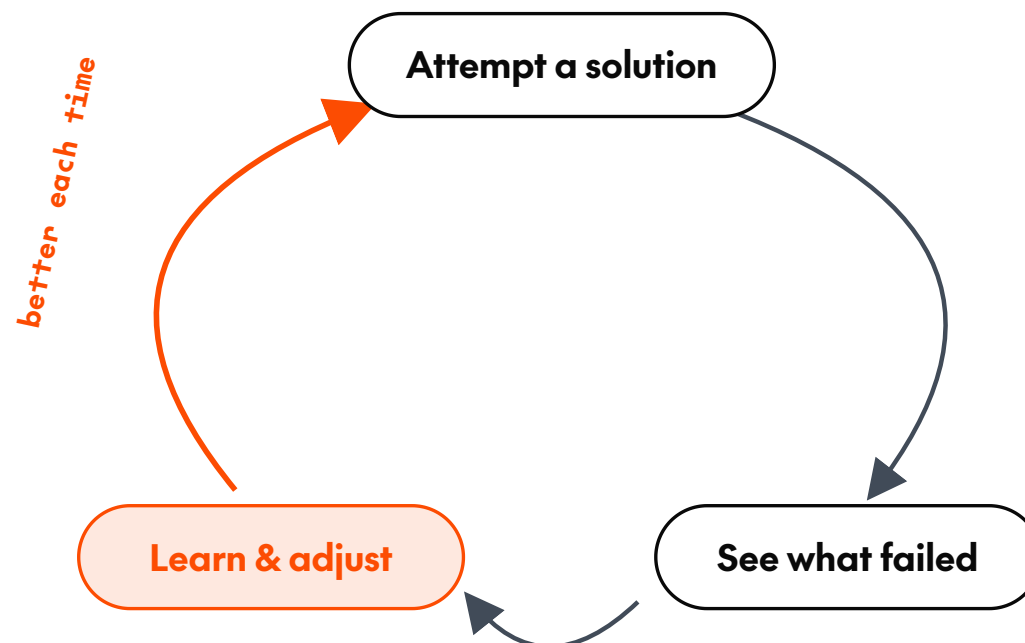


And it keeps trying the **same thing** over and over.

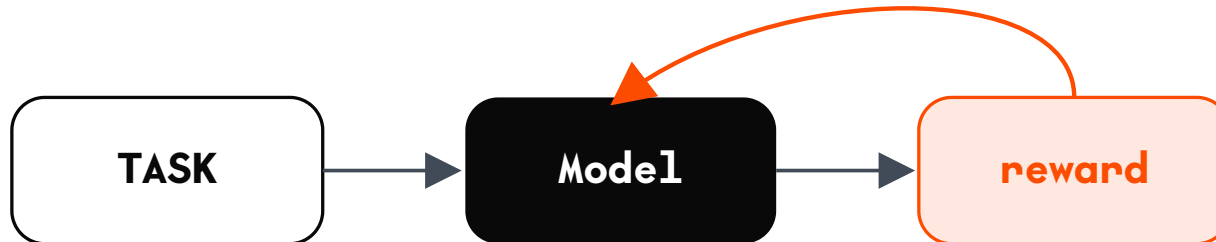


The model never learns from its failures. It keeps trying the **same bad ideas**.

No one solves a hard problem on the **first try**.

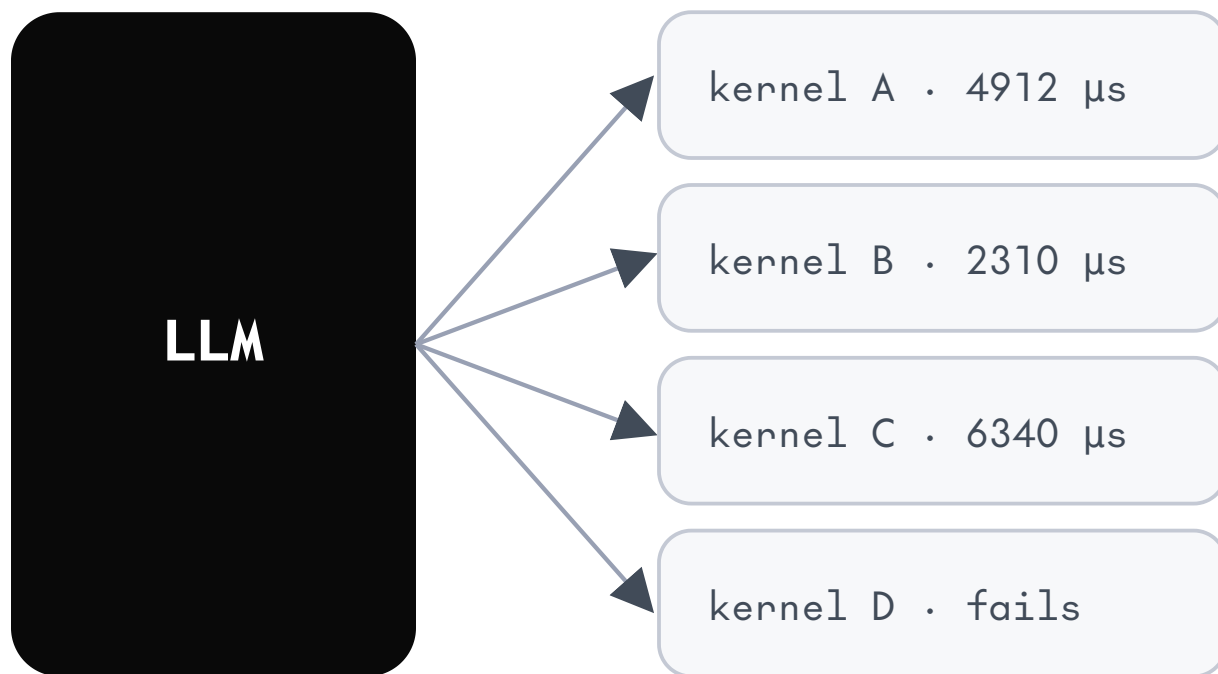


We can just **train on the same task**, over and over, at test time.

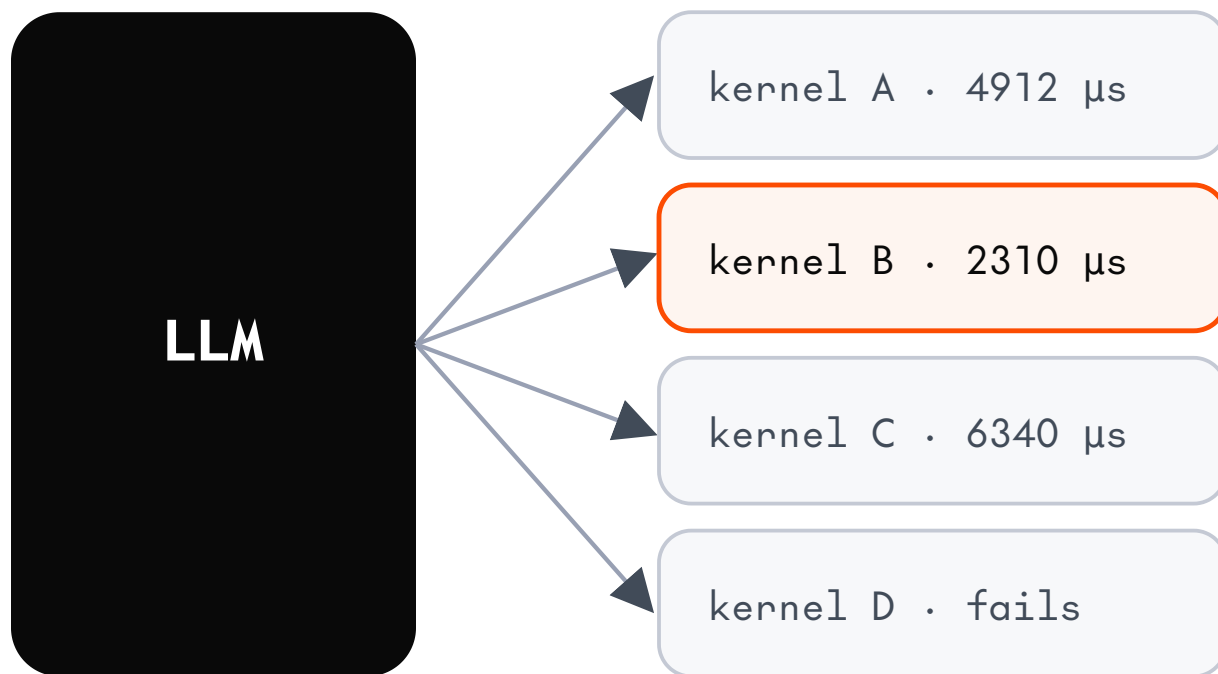


Let's do **RL**.

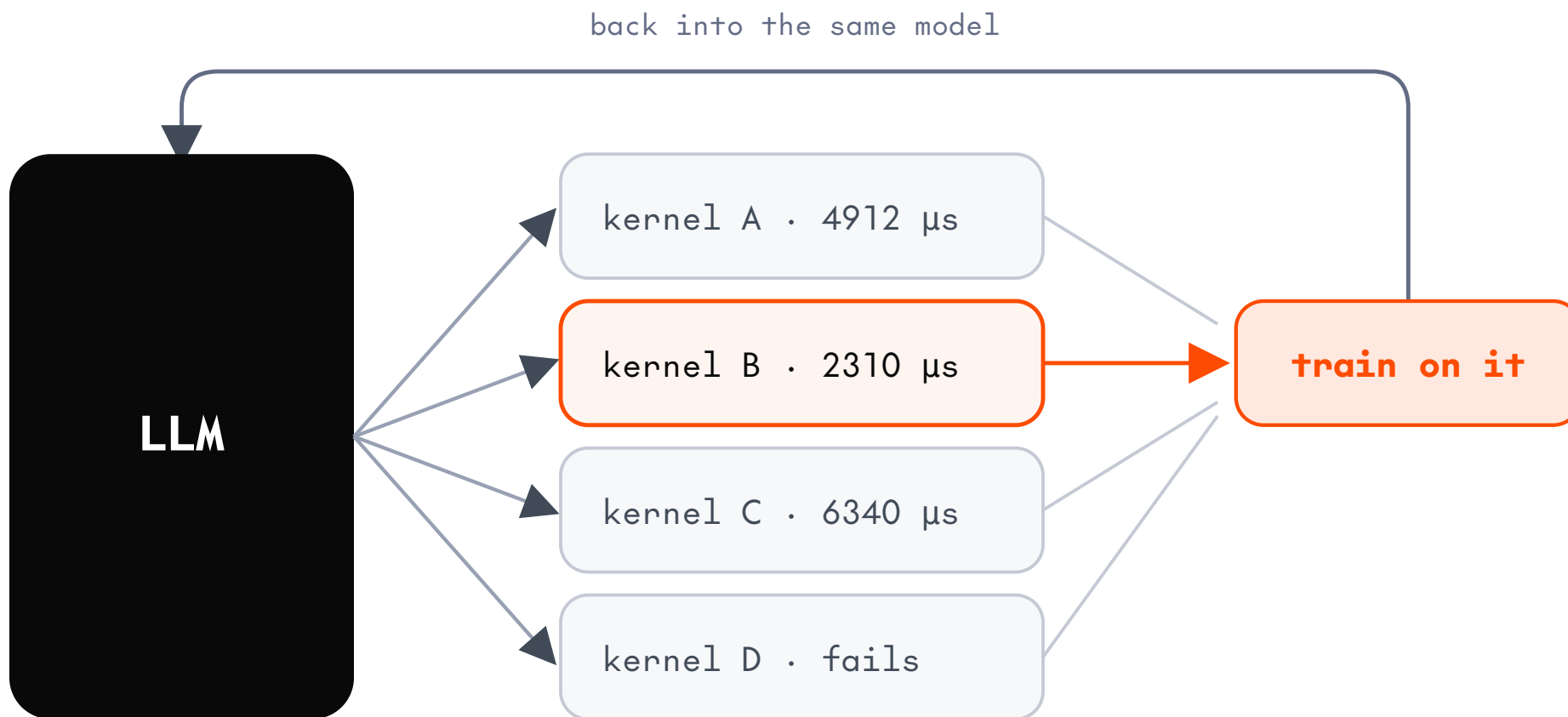
Refresher: what does **training** mean?



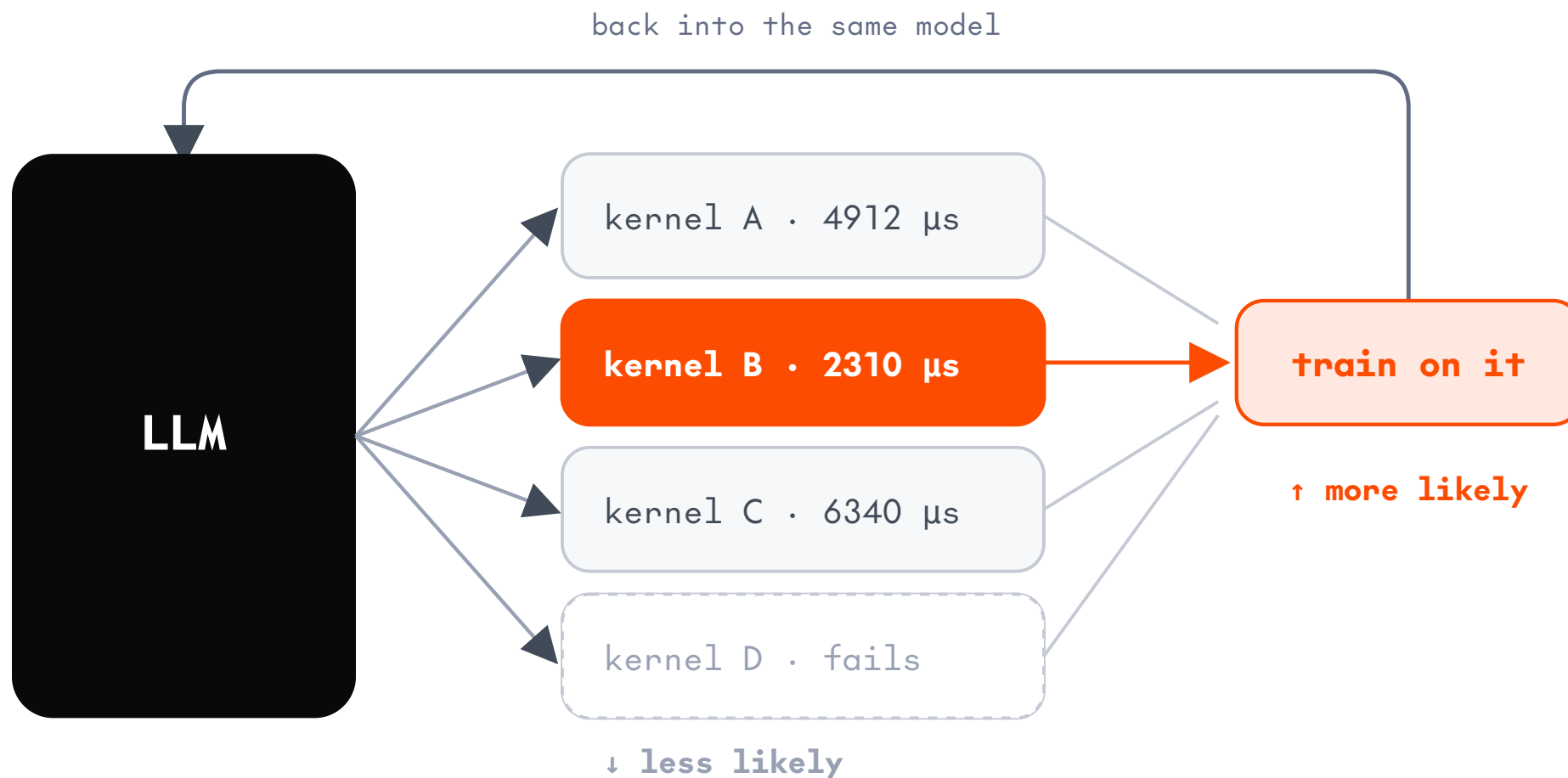
Refresher: what does **training** mean?



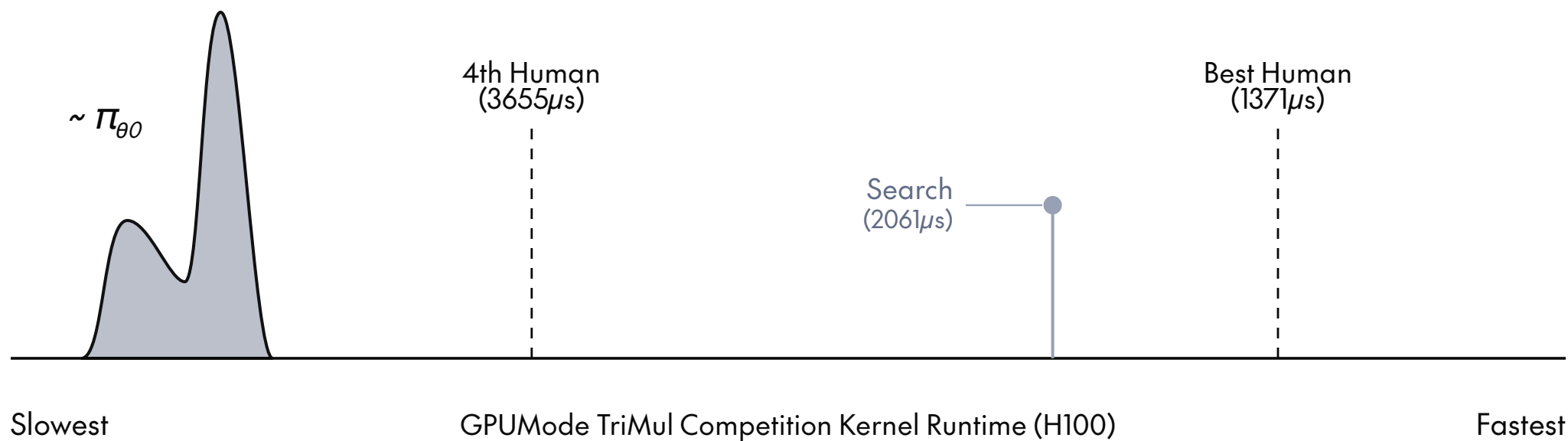
Refresher: what does **training** mean?



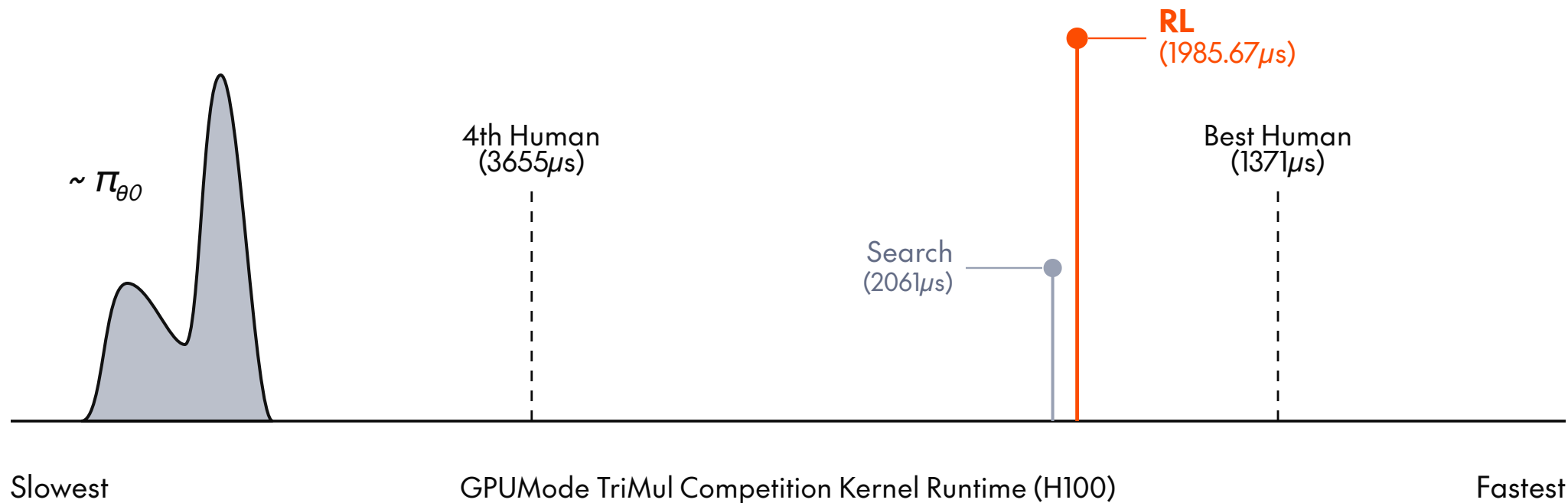
Refresher: what does **training** mean?



RL is **better** but not human level yet



RL is **better** but not human level yet

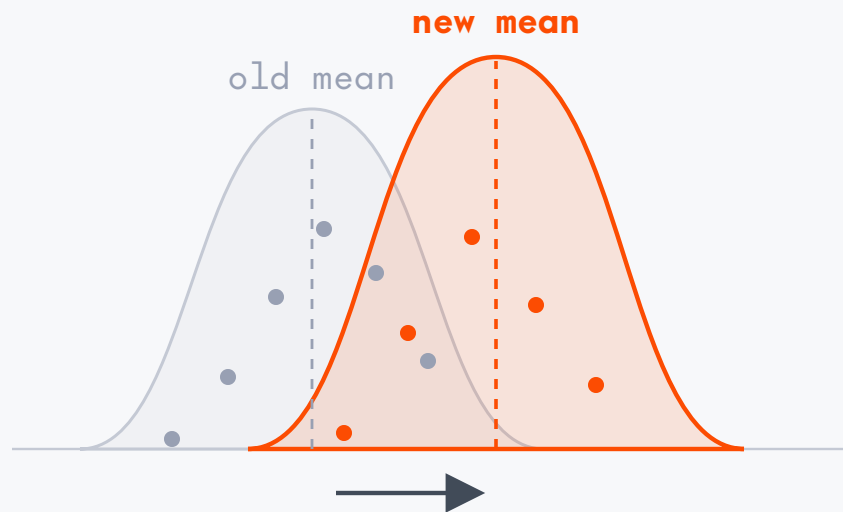


Why can't standard RL bring us **there**?

Why can't standard RL bring us **there**?

STANDARD RL

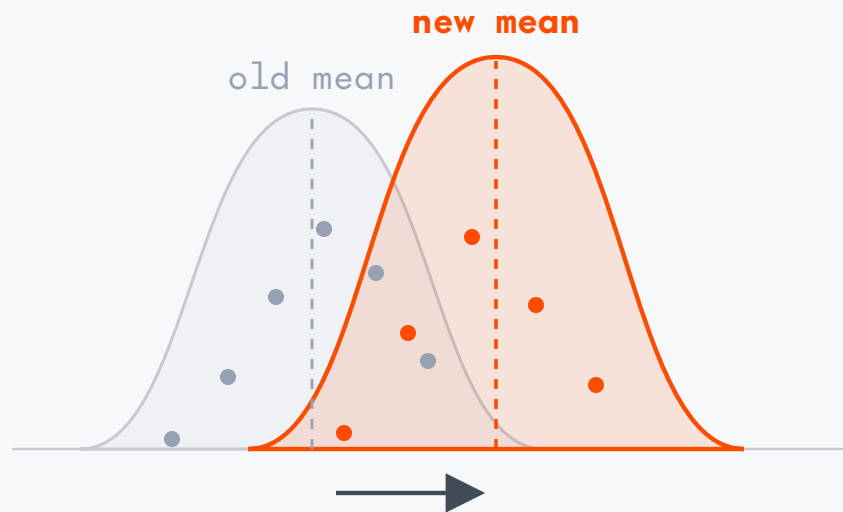
raise the typical rollout



Why can't standard RL bring us **there**?

STANDARD RL

raise the typical rollout



DISCOVERY

we only keep the best rollout

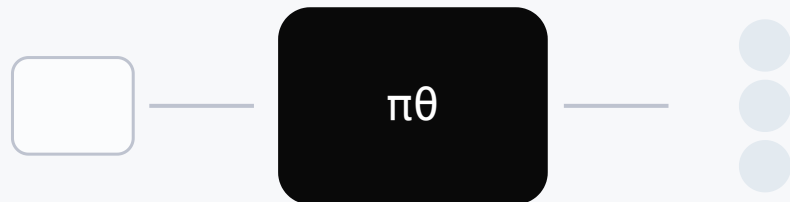
record solution



The **artifact** is the goal of the discovery process.

The **artifact** is the goal of the discovery process.

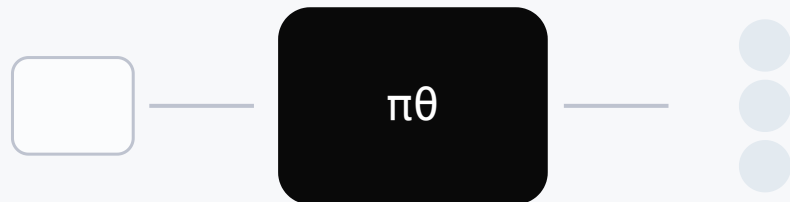
STANDARD RL
keep the **policy**



the policy is the product

The **artifact** is the goal of the discovery process.

STANDARD RL
keep the **policy**



the policy is the product

TTT-DISCOVER
keep the **best solution**



the artifact is the product

Reinforcement learning at test time,
on a single problem,
trained to maximize the best sample,
not the average.

One edit to RL to guide learning to discover

Optimize the **best** sample, not the average.

$$\mathbb{E}[R] \rightarrow \log \mathbb{E}[e^{\beta R}]$$

From average reward to **best-sample reward**.

STANDARD RL

maximize

$$\mathbb{E}_{y \sim \pi_{\theta}} [R(y)]$$

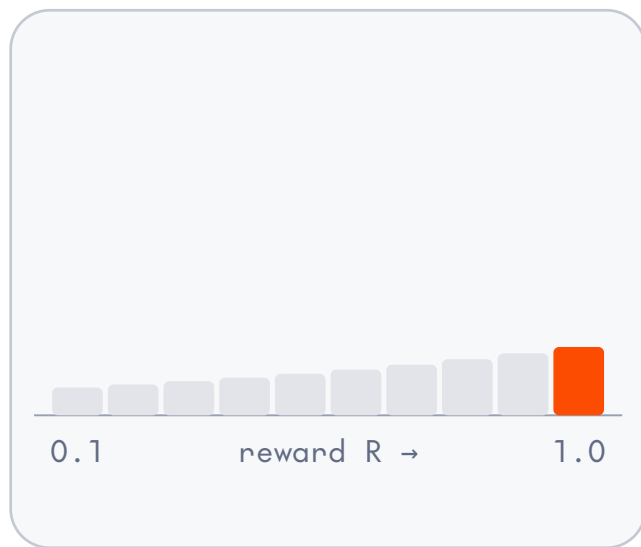
TTT-DISCOVER

maximize

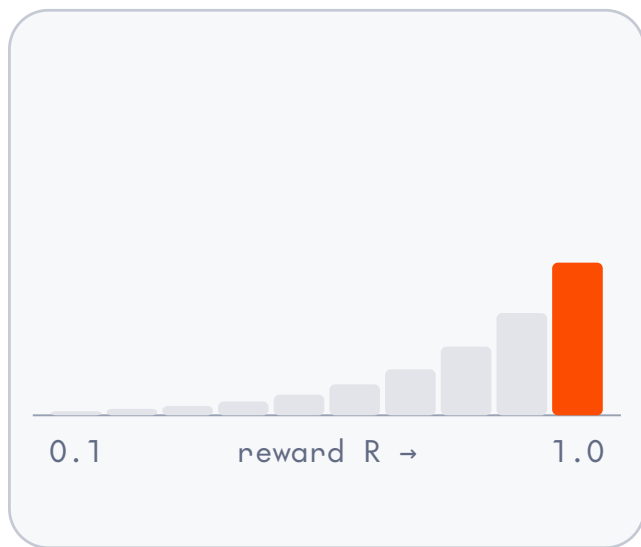
$$\log \mathbb{E}_{y \sim \pi_{\theta}} \left[e^{\beta R(y)} \right]$$

y = a sampled solution • $R(y)$ = verifier score

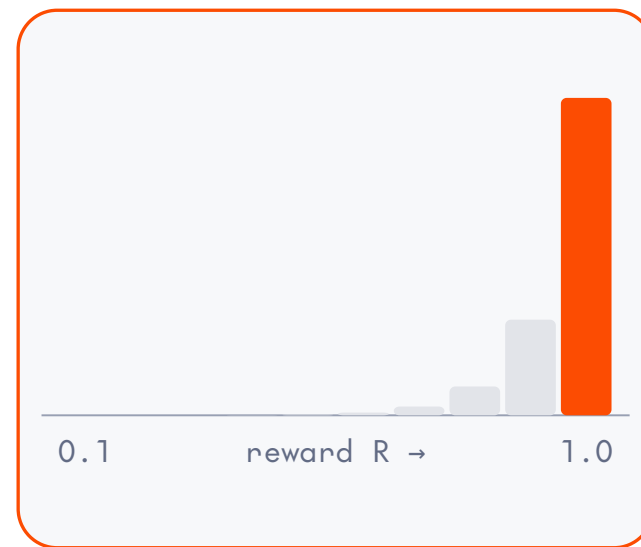
Higher β means the weight will pile on the **best** rollout

 $\beta = 1$ 

close to uniform

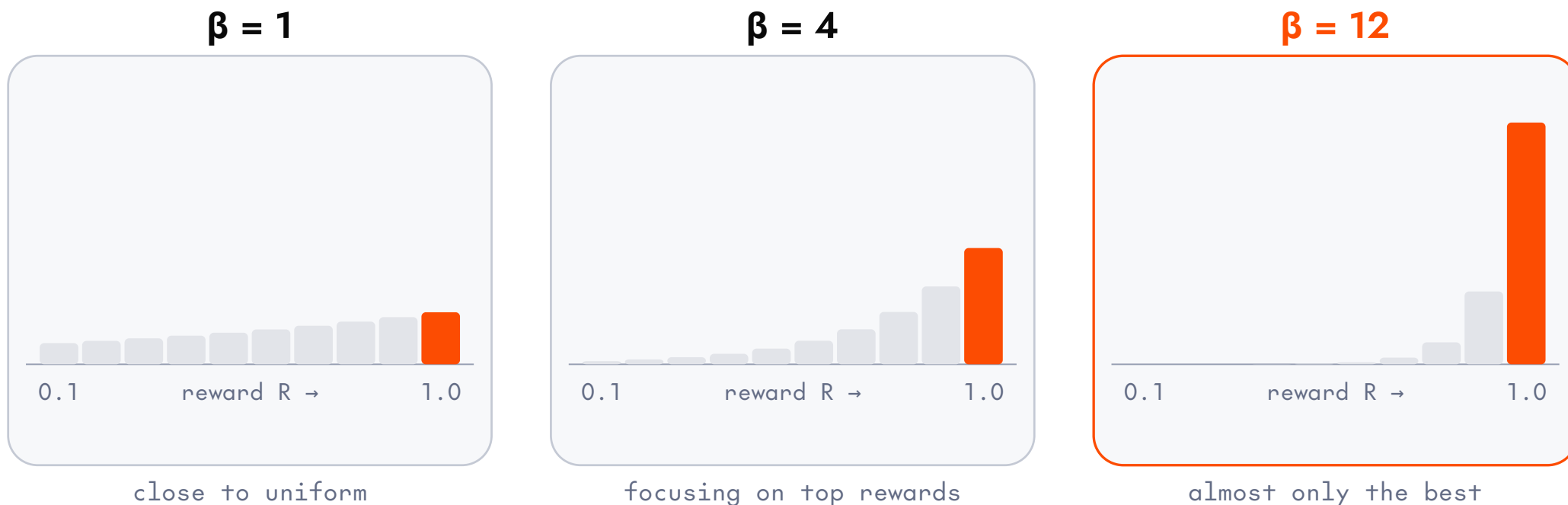
 $\beta = 4$ 

focusing on top rewards

 $\beta = 12$ 

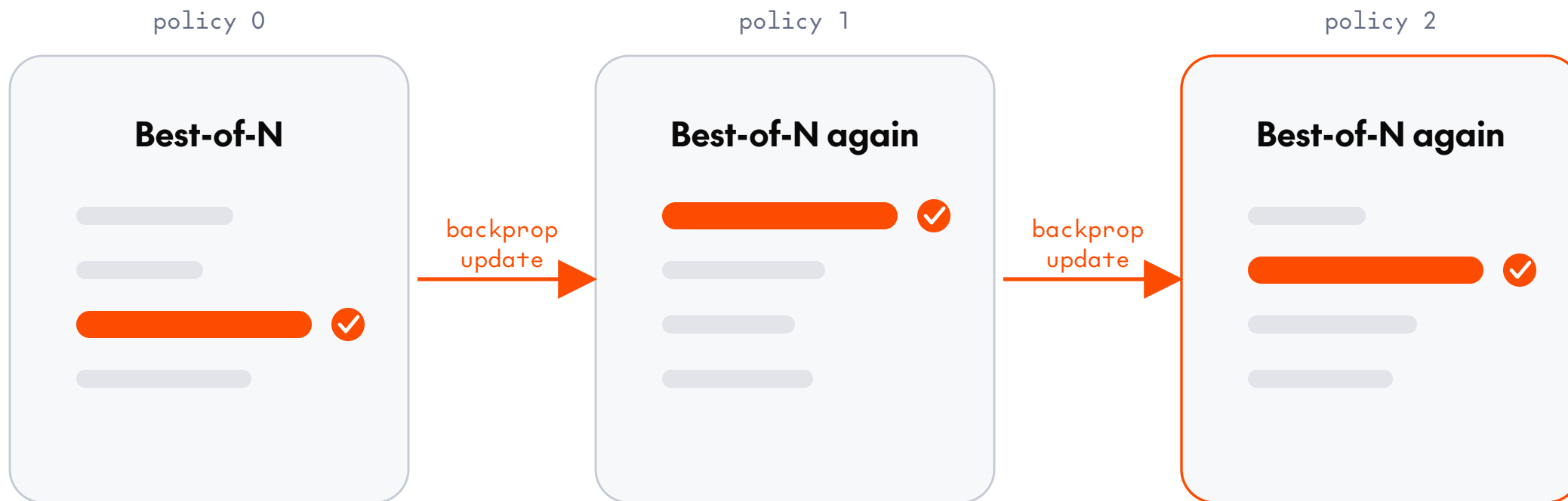
almost only the best

Higher β means the weight will pile on the **best** rollout

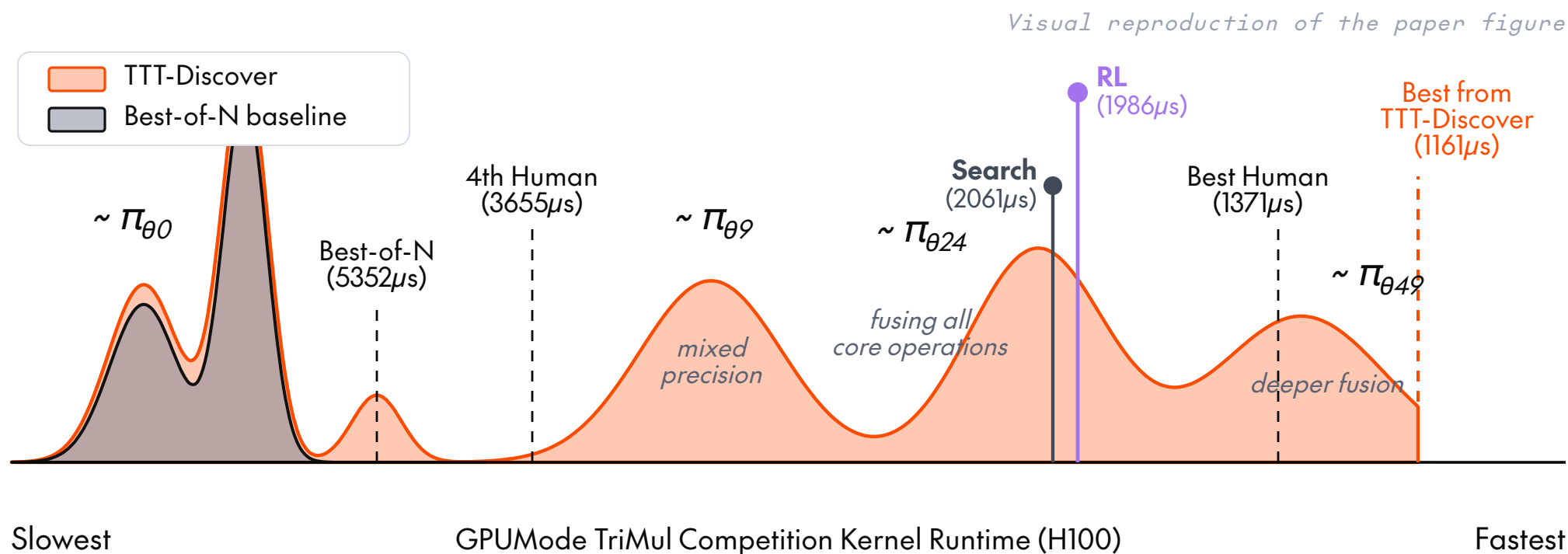


A constant β (from the literature) stalls late in training. **Our contribution:** adapting β with a KL budget is what keeps improving.

Updates **shift** the sampling distribution.



Each update shifts samples toward **faster** kernels.



TTT-Discover in a few lines of code.

```
discover.py

from ttt_discover import Environment, RewardEvaluator, DiscoverConfig, discover

# 1. score a candidate
class YourReward(RewardEvaluator):
    def get_reward(self, code, state):
        ...
        return {"reward": reward, "correctness": 1.0}

# 2. describe the task
class YourEnv(Environment):
    reward_function = YourReward
    def get_question(self):
        return f"You are an expert ... {self.initial_state.to_prompt()}"

# 3. run it
discover(DiscoverConfig(env_type=YourEnv, experiment_name="test-run"))
```

Results

We apply this to kernel engineering, mathematics, algorithm design and denoising.

The training setup.

BASE MODEL

gpt-oss-120b

FINE-TUNING

LoRA

ALGORITHM

Entropic objective

Six results in four different domains.

problem

previous record

TTT-Discover

Six results in four different domains.

problem	previous record	TTT-Discover
TriMul kernel · H100	1371 μ s	1161 μs

Six results in four different domains.

problem	previous record	TTT-Discover
TriMul kernel · H100	1371 μ s	1161 μs
Erdős minimum overlap	0.380924	0.380876

Six results in four different domains.

problem	previous record	TTT-Discover
TriMul kernel · H100	1371 μ s	1161 μs
Erdős minimum overlap	0.380924	0.380876
Autocorrelation C_1	1.50314	1.50287

Six results in four different domains.

problem	previous record	TTT-Discover
TriMul kernel · H100	1371 μ s	1161 μs
Erdős minimum overlap	0.380924	0.380876
Autocorrelation C_1	1.50314	1.50287
AtCoder AHC039	566,997	567,062

Six results in four different domains.

problem	previous record	TTT-Discover
TriMul kernel · H100	1371 μ s	1161 μs
Erdős minimum overlap	0.380924	0.380876
Autocorrelation C_1	1.50314	1.50287
AtCoder AHC039	566,997	567,062
AtCoder AHC058	848,373,282	848,414,228

Six results in four different domains.

problem	previous record	TTT-Discover
TriMul kernel · H100	1371 μ s	1161 μs
Erdős minimum overlap	0.380924	0.380876
Autocorrelation C_1	1.50314	1.50287
AtCoder AHC039	566,997	567,062
AtCoder AHC058	848,373,282	848,414,228
Single-cell denoising	0.64	0.71

Six results in four different domains.

problem	previous record	TTT-Discover
TriMul kernel · H100	1371 μ s	1161 μs
Erdős minimum overlap	0.380924	0.380876
Autocorrelation C_1	1.50314	1.50287
AtCoder AHC039	566,997	567,062
AtCoder AHC058	848,373,282	848,414,228
Single-cell denoising	0.64	0.71

We asked **domain experts** to review every result.

Kernel engineering

Kernel engineering: **task**

THE TASK

A GPU kernel for the triangle-multiplicative update, a core op of AlphaFold 3.

WHAT WE WANT

A kernel faster than the human baseline.

$$z_{ij} = \text{LayerNorm}\left(\sum_k a_{ik} \odot b_{jk}\right)$$

TO BEAT

best human · GPU MODE

1371

μs · lower is better

Kernel engineering: results

```
trimul.py 470 lines · Triton · condensed

# fused projection + gating + mask
@triton.jit
def _proj_gate_mask_kernel(x_ptr, mask_ptr,
    left_proj_w_ptr, left_gate_w_ptr, ...):
    acc_lp = tl.zeros((BLOCK_M, BLOCK_H), tl.float32)
    acc_lg = tl.zeros((BLOCK_M, BLOCK_H), tl.float32)
    for k in range(0, C, BLOCK_K):
        a = tl.load(x_ptr + ...,
            mask=row_mask[:, None] & k_mask[None, :])
        w_lp = tl.load(left_proj_w_ptr + ...)
        acc_lp += tl.dot(a, w_lp) # 5 projections,
        acc_lg += tl.dot(a, w_lg) # one pass over x
# fused LayerNorm -> out-gate -> final linear
@triton.jit
def _ln_gate_out_linear_fused_kernel(...):
```

best human · GPU MODE

1371 μ s

TTT-Discover

1161 μ s

H100 · lower is better

What the model **added** to the kernel

What the model **added** to the kernel

1 **Saw the bottleneck**

What the model **added** to the kernel

1 Saw the bottleneck

2 Fused everything around them

What the model **added** to the kernel

1 Saw the bottleneck

2 Fused everything around them

EXPERT REVIEW

“Overall the agent’s strategy is to reduce memory bandwidth via fusions, lower precision and delegating the big matrix multiplications to cuBLAS, as those are non-trivial to beat. This is similar to the current best human solutions, but executed on better. Most of the human solutions lag behind in fusing some of the more complex operators together, resulting in this solution outperforming them by a large margin.”

GPU MODE organizer

The TriMul kernel generalizes also to other chips

GPU KERNELS · TRIMUL

runtime μ s · lower is better

1st human

TTT-Discover

reduction



2.06x

faster on A100 · -51%

Trained on H100 · tested on A100, H100, B200, MI300X

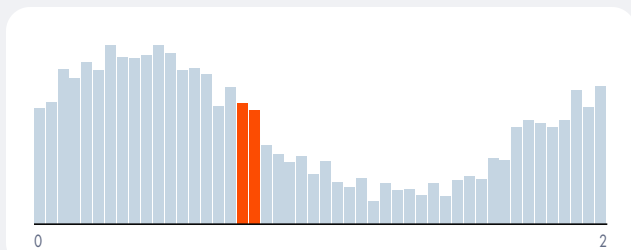
Mathematics

In these problems the discovery is often an **array**.

In these problems the discovery is often an **array**.

1 · THE ANSWER

A construction, stored as a list of numbers.

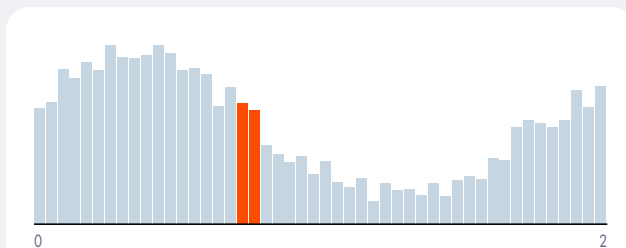


$x = [0.12, 0.31, 0.94, 0.97, 0.40, \dots]$

In these problems the discovery is often an **array**.

1 · THE ANSWER

A construction, stored as a list of numbers.



`x = [0.12, 0.31, 0.94, 0.97, 0.40, ...]`

2 · WHAT THE MODEL WRITES

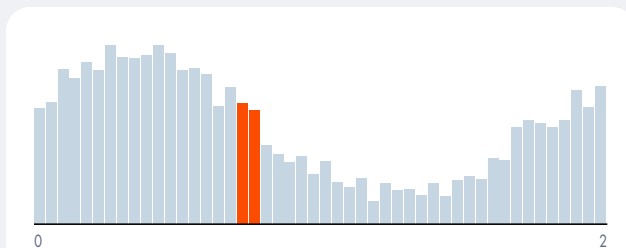
A program that improves the array.

```
def search(x, budget):  
    while time() < budget:  
        x2 = perturb(x)  
        if score(x2) > score(x): x = x2  
    return x
```

In these problems the discovery is often an **array**.

1 . THE ANSWER

A construction, stored as a list of numbers.



`x = [0.12, 0.31, 0.94, 0.97, 0.40, ...]`

2 . WHAT THE MODEL WRITES

A program that improves the array.

```
def search(x, budget):  
    while time() < budget:  
        x2 = perturb(x)  
        if score(x2) > score(x): x = x2  
    return x
```

3 . THE REWARD

The bound the returned array achieves, checked by a verifier.

$$R(x) = -\text{bound}(x)$$

smaller bound $\rightarrow$ higher reward

Minimum overlap: task

THE TASK

Choose a density f on $[0, 2]$ to minimise the worst-case overlap with its shift.

WHAT WE WANT

A bound tighter than AlphaEvolve's.

TO BEAT

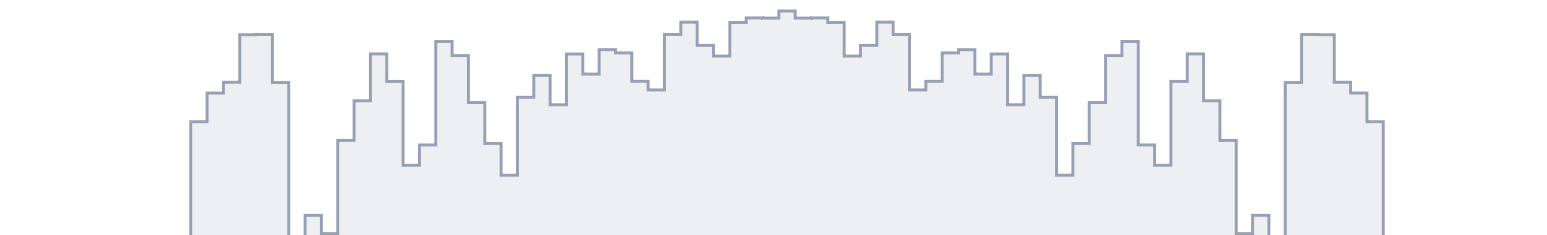
AlphaEvolve

0.380924

lower is better

Minimum overlap: results

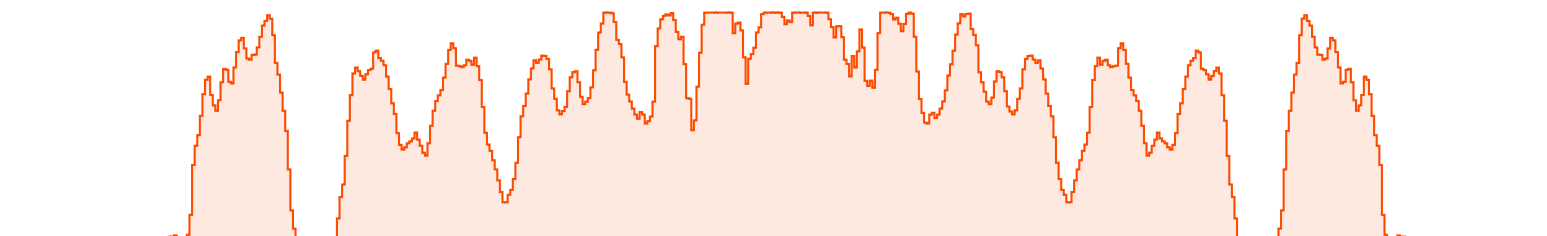
ALPHAEVOLVE · 95 PIECES · SYMMETRIC



AlphaEvolve

0.380924

TTT-DISCOVER · 600 PIECES · ASYMMETRIC



TTT-Discover

0.380876

A step **16×** larger than
AlphaEvolve's.

Autocorrelation inequality: task

THE TASK

Find a non-negative f whose autoconvolution $f * f$ has the smallest possible peak.

WHAT WE WANT

A bound tighter than ThetaEvolve's.

TO BEAT

ThetaEvolve

1.50314

lower is better

Autocorrelation inequality: results

AlphaEvolve · DeepMind	1.50530
AlphaEvolve V2	1.50317
ThetaEvolve	1.50314

TTT-Discover

1.50287

ThetaEvolve

1.50314

TTT-Discover

1.50287

What the model **added** to the Erdős search

What the model **added** to the Erdős search

1 **FFT-accelerated gradient descent**

What the model **added** to the Erdős search

1 FFT-accelerated gradient descent

2 Hill climbing and annealing on top

What the model **added** to the Erdős search

1 **FFT-accelerated gradient descent**

2 **Hill climbing and annealing on top**

EXPERT REVIEW

“It is straightforward to verify that the provided functions give bounds that improve upon the state of the art.”

Prof. Davide Torlo · Università di Roma La Sapienza

Algorithm design

`</>` Purse seine fishing: **task**

THE TASK

Draw an axis-aligned net of bounded length that catches mackerel and avoids sardines.

WHAT WE WANT

A program that scores above the leaderboard.

TO BEAT best human **566,997** contest score · higher is better

Purse seine fishing: results

```
ahc039.cpp 856 lines · C++ · condensed

// the net = union of axis-aligned rectangles
struct SimpleRect { int L, R, B, T; };
// O(1) mackerel - sardine count via prefix sums
int weightRect(int L, int R, int B, int T) {
    int a = pref[prefIdx(iR + 1, iT + 1)];
    int b = pref[prefIdx(iL, iT + 1)];
    int c = pref[prefIdx(iR + 1, iB)];
    int d = pref[prefIdx(iL, iB)];
    return a - b - c + d;
}
// simulated annealing over the rectangle set
int deltaAdd(const SimpleRect& R);
int deltaRemove(int idx);
SimpleRect kadaneRect(); // best sub-rectangle
```

best human · contest

566,997

TTT-Discover

567,062

AtCoder official score · higher is better

`</>` Production scheduling: **task**

THE TASK

Machines build machines that make apples;
choose the upgrade order to maximise output in
500 steps.

WHAT WE WANT

A program that scores above the leaderboard.

TO BEAT

ALE-Agent

848,373,282

contest score · higher is better

Production scheduling: results

```
ahc058.cpp 610 lines · C++ · condensed

/* net gain = future apples from upgrade */
inline __int128 net_gain(const State &s, int steps,
                        int lvl, int id) {
    __int128 gain = marginal_gain(s, steps, lvl, id);
    if (lvl == 0) gain += (__int128)A[id] * s.B[0][id];
    return gain;
}

/* beam search over the first 250 steps */
constexpr int EARLY_STEPS = 250, BEAM_SIZE = 600;
/* simulated annealing on the full 500-step plan */
if (op < 30) { // random changes }
else if (op < 60) { // swap two }
else if (op < 80) { // move }
else if (op < 98) { // replace segment / suffix with greedy }
```

ALE-Agent · Sakana · 1st place

848,373,282

TTT-Discover

848,414,228

AtCoder official score · higher is better

What the model **added** to the programs

What the model **added** to the programs

1

The program scores candidates in constant time with prefix sums

What the model **added** to the programs

1

The program scores candidates in constant time with prefix sums

2

It builds a pool of good starting plans before annealing

Biology

Single-cell denoising: **task**

THE TASK

Recover true gene expression from sparse, noisy single-cell counts.

$$\max_{\text{denoise}} \text{score}(\text{denoise}(X_{\text{obs}}), X_{\text{true}})$$

WHAT WE WANT

A denoiser that scores above MAGIC.

TO BEAT

MAGIC

0.64

Open Problems score · higher is better

Single-cell denoising: results

denoise_+++.py 735 lines · Python · condensed

```
def _adaptive_blend_weights(dropout, var_orig,
    var_diff, corr, mu, max_alpha=0.55, eps=1e-12):
    """Diffusion-blend weight for each gene."""
    var_reduction = (var_orig - var_diff) / (var_orig + eps)
    var_reduction = np.clip(var_reduction, 0.0, 1.0)
    mu_norm = (mu - mu.min()) / (mu.max() - mu.min() + eps)
    expr_factor = 1.0 - mu_norm
    raw = dropout * var_reduction * (1.0 - corr) * expr_factor
    raw = np.where(dropout > 0.8, raw * 1.2, raw)
    return np.clip(raw, 0.0, max_alpha)
# ---- blend raw and diffused signals
blended = X_norm * (1.0 - w_diff) + diffused * w_diff
```

MAGIC
0.64

TTT-Discover
0.71
Open Problems score · higher is better

What the model **added** to MAGIC

What the model **added** to MAGIC

1 **Gene-adaptive blending**

What the model **added** to MAGIC

1 Gene-adaptive blending

2 Low-rank SVD refinement

What the model **added** to MAGIC

1 Gene-adaptive blending

2 Low-rank SVD refinement

3 Log-space polishing

What the model **added** to MAGIC

1 Gene-adaptive blending

2 Low-rank SVD refinement

3 Log-space polishing

EXPERT REVIEW

“The proposed improvement on the MAGIC algorithm is simple [...] and yields empirical improvements on key metrics. However, improvements on metrics [...] may not always transfer to enhanced ability to obtain new biological insights.”

Prof. Eric Sun · MIT

For some tasks we didn't find a **new discovery**

Second autocorrelation
inequality

Looser bound than the record.

TTT-Discover	AlphaEvolve V2
0.959	0.961

Circle packing

Matched the record. No
improvement.

TTT-Discover	best known
matched	n = 26

MLA-Decode kernel

Within noise of the top human
on MI300X.

TTT-Discover	best human
1669 μs	1654 μs

Limitations

Limitations

Works only on verifiable tasks

Better if the verifier is efficient.

Limitations

Works only on verifiable tasks

Better if the verifier is efficient.

Works only on continuous rewards

Pass or fail gives nothing to climb.

Observations

Three things we noticed along the way.

The model learns to write kernels that **actually run**.

KERNELS THAT PASS CORRECTNESS CHECKS

80%

60%

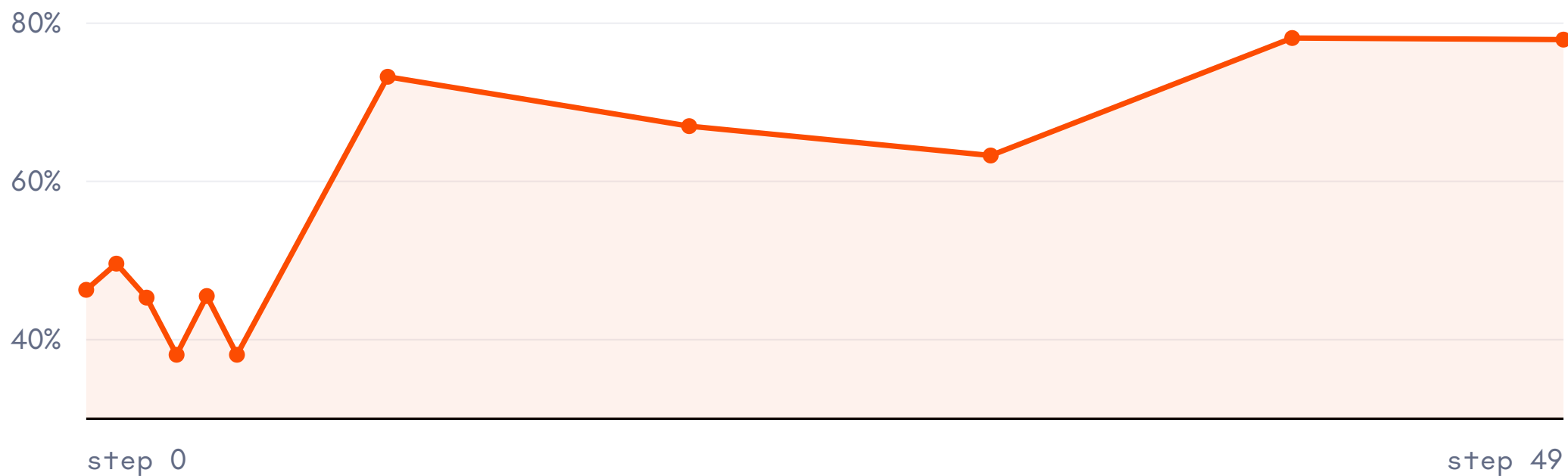
40%

step 0

step 49

The model learns to write kernels that **actually run**.

KERNELS THAT PASS CORRECTNESS CHECKS



Can **small models** discover?

BEAT THE BEST
HUMAN?

best human

Minimum overlap

Erdős bound · lower
is better

0.380927

TriMul kernel

μs · lower is better

1371

Can **small models** discover?

BEAT THE BEST
HUMAN?

best human

Qwen3-4B

Minimum overlap

Erdős bound · lower
is better

0.380927

0.381115 ✕

TriMul kernel

μs · lower is better

1371

9726 ✕

Can **small models** discover?

BEAT THE BEST
HUMAN?

best human

Qwen3-4B

Qwen3-8B

Minimum overlap

Erdős bound · lower
is better

0.380927

0.381115 ✕

0.380920 ✓

TriMul kernel

μs · lower is better

1371

9726 ✕

9729 ✕

Can **small models** discover?

BEAT THE BEST
HUMAN?

best human

Qwen3-4B

Qwen3-8B

gpt-oss-120b

Minimum overlap

Erdős bound · lower
is better

0.380927

0.381115 ✕

0.380920 ✓

0.380876 ✓

TriMul kernel

μs · lower is better

1371

9726 ✕

9729 ✕

1161 ✓

Does the model forget?

Does the model **forget**?

We train on one problem, then score the checkpoint on a different one, AC1.

Does the model **forget**?

We train on one problem, then score the checkpoint on a different one, AC1.

base model

1.51504

Does the model **forget**?

We train on one problem, then score the checkpoint on a different one, AC1.

base model

1.51504

after training on Erdős

1.51234

Does the model **forget**?

We train on one problem, then score the checkpoint on a different one, AC1.

base model

1.51504

after training on Erdős

1.51234

after training on TriMul

1.51296

Conclusion

Learning to **discover** kernels at test time

Learning to **discover** kernels at test time

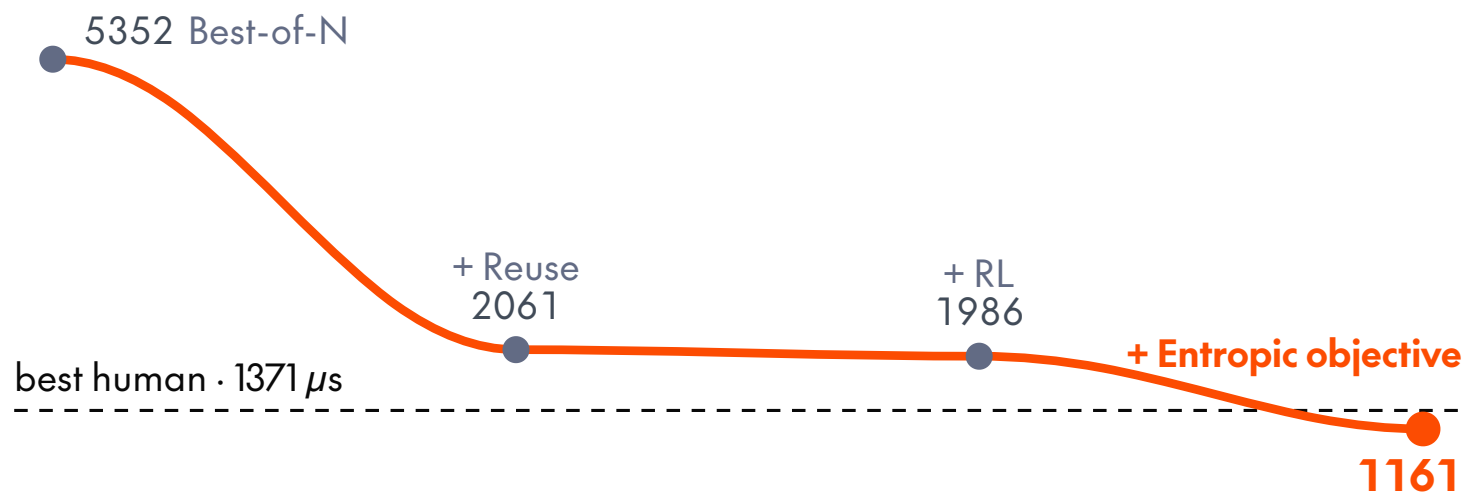
Best-of-25,600 samples

5352 μ s

Learning to **discover** kernels at test time

Best-of-25,600 samples

5352 μs



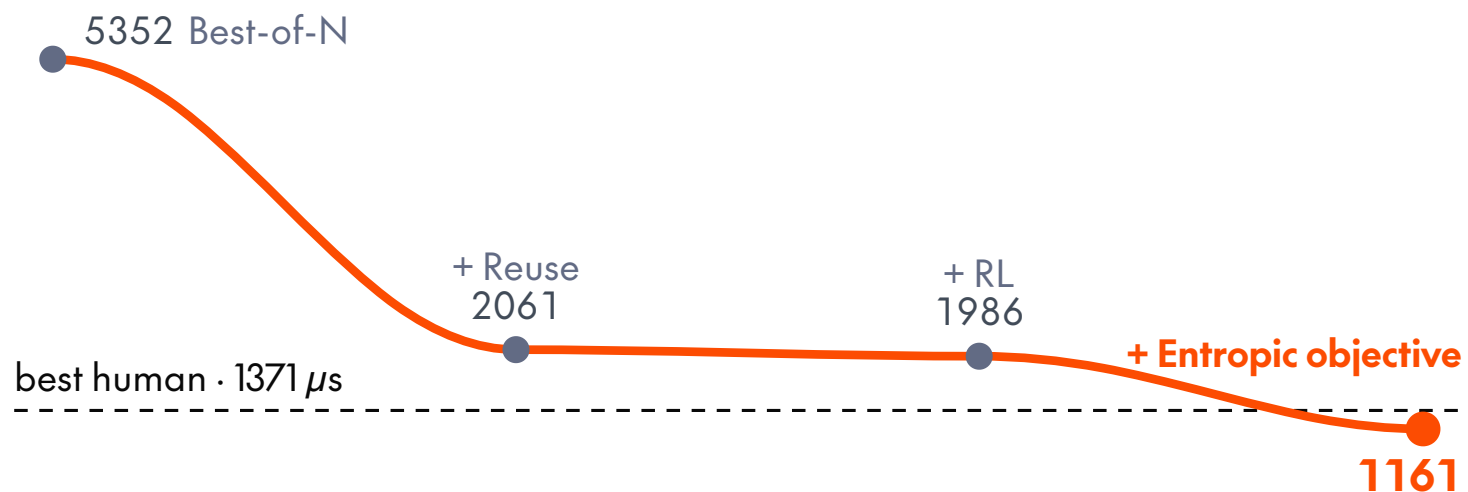
Learning to **discover** kernels at test time

Best-of-25,600 samples

5352 μs

TTT-Discover

1161 μs



NORMAL RL

Rewards a policy for doing well on average, because it gets deployed again and again.

NORMAL RL

Rewards a policy for doing well on average, because it gets deployed again and again.



NORMAL RL

Rewards a policy for doing well on average, because it gets deployed again and again.



DISCOVERY

Needs **one good rollout**. One kernel, one bound, one algorithm that becomes the new state of the art.

NORMAL RL

Rewards a policy for doing well on average, because it gets deployed again and again.



DISCOVERY

Needs **one good rollout**. One kernel, one bound, one algorithm that becomes the new state of the art.

Some discoveries can be made by
learning at test time.

