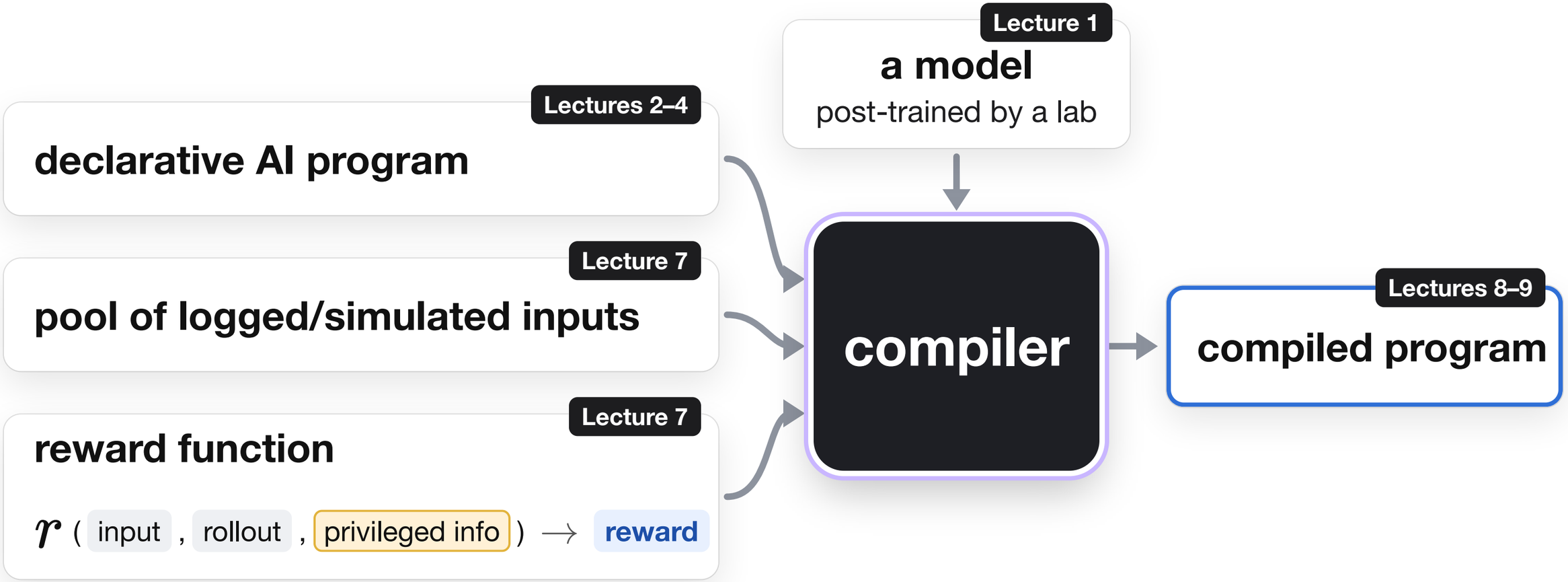
Downstream Learning for AI Systems

Omar Khattab

Engineering AI Systems and Agents



it learns…

- ✗ our top-level requirements
- ✓ an optimized implementation: mid-level hypotheses and low-level patches

it needs to be autonomous, to…

- ✗ always beat hand-crafted systems
- ✓ be maintainable: keep up as models and program structure change

How should we write **durable** AI programs?

1 · Symbolic code for composition and controlled information flow.

2 · Typed natural-language signatures for semantic work.

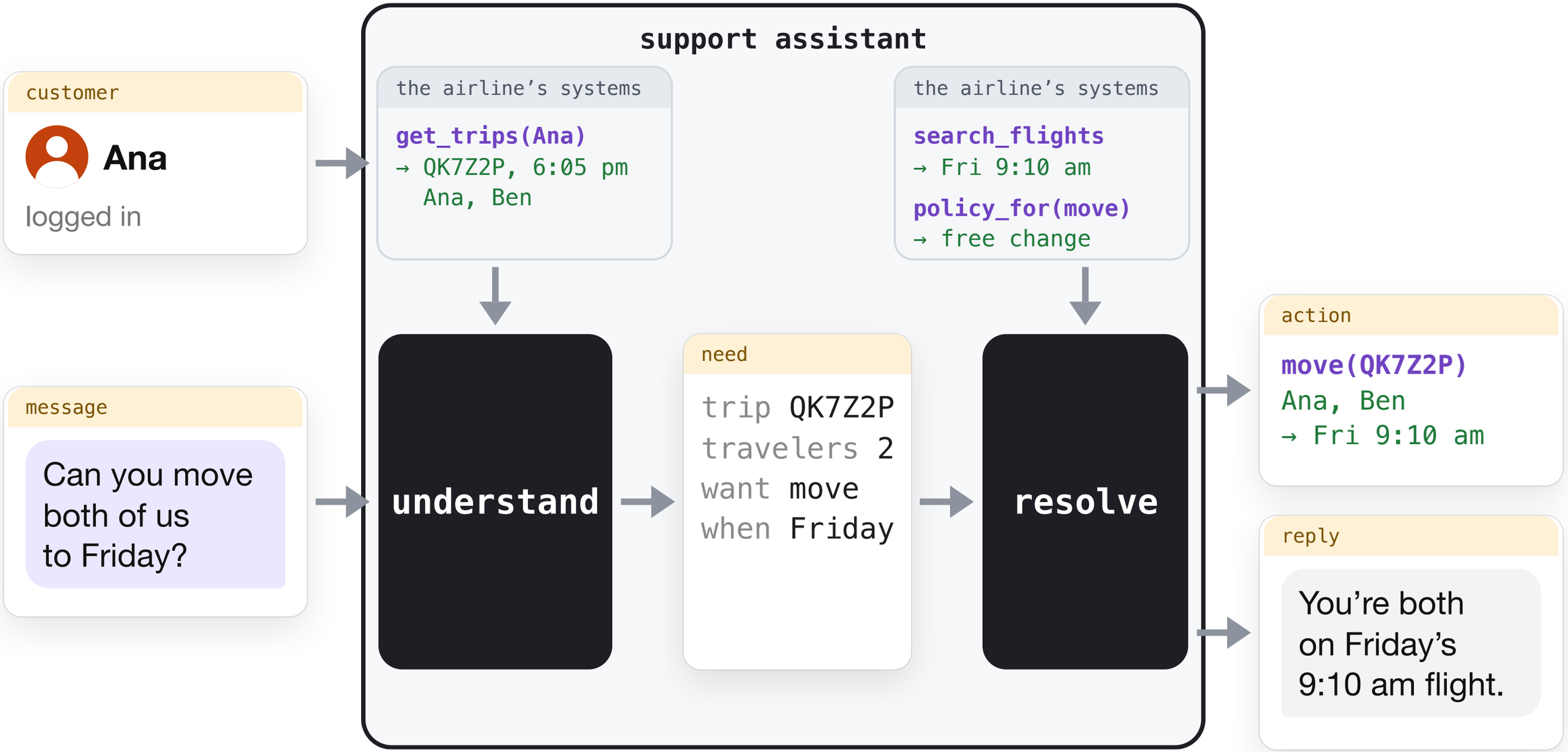
3 · Data or feedback for the distribution of inputs, and for edge cases that are hard to specify otherwise!

```python
def support_assistant(customer, message):
    trips = get_trips(customer)
    need = understand(message, trips)
    flights = search_flights(need.trip, need.when)
    policy = policy_for(need.want)
    action, reply = resolve(message, need, flights, policy)
    run(action)
    return reply
```

```python
class Understand(Signature):
    """Find who and what they want."""
    message: str = InputField()
    trips: list[Trip] = InputField()
    need: Need = OutputField(
        desc="trip, travelers, want, when")
```

How should we fill these two boxes?

Recall our airline support assistant from Lecture 7



In today's lecture, we will talk about learning through **text optimization**, operated by a language model!



What would it take for this to work?

Approximation
Does there exist a program, just words, that does our task well?

Optimization
If one exists, can we find it from a bunch of training requests?

Generalization
Does the program we found work well on unseen requests?

Can we deploy this pipeline?

We designed a good pipeline, and the model ought to be good enough at this. 1

K

Lecture 7

I tried it a bunch, and it seems pretty good. 2

Deploying based on 2 assumes

A enough tries looked good

B I tried the right requests

C next month is like this month

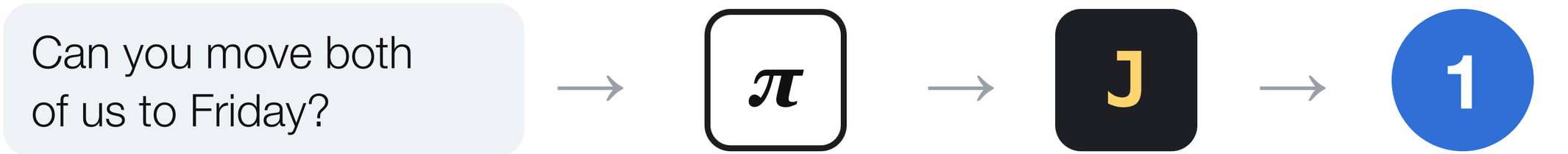
What can evaluation scores guarantee?

Today

1. What can evaluation scores guarantee?
2. Learning from trial and error
3. Learning from search
4. Learning from reflection

Part 1

What can evaluation scores guarantee?



$$S(\pi) = \frac{1}{300} \sum_{i=1}^{300} r(x_i, \tau_i) = 86\%$$

its score: the average reward on 300 requests, sampled D at random

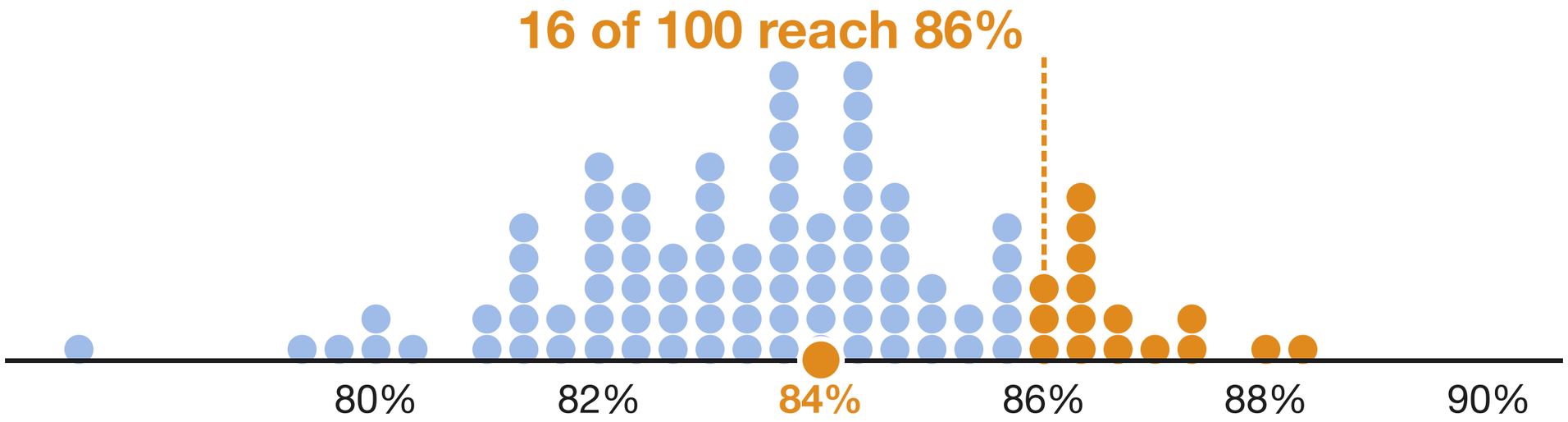
$$A(\pi) = \mathbb{E}_{x \sim \mathcal{D}} \; \mathbb{E}_{\tau \sim \pi(x)} \left[r(x, \tau) \right] = \textbf{?}$$

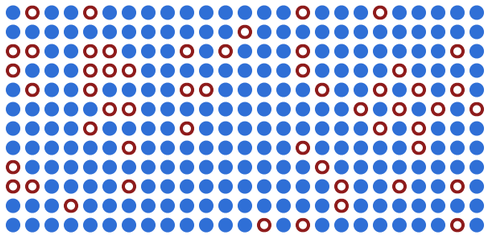
its actual expected reward, over requests and its rollouts

How low can *A*(*π*) be?

One reward is one flip of a coin: **1** chance $A(\pi)$ **0** chance $1 - A(\pi)$

If $A(\pi)$ were only 84%, could 300 flips still reach 86%?



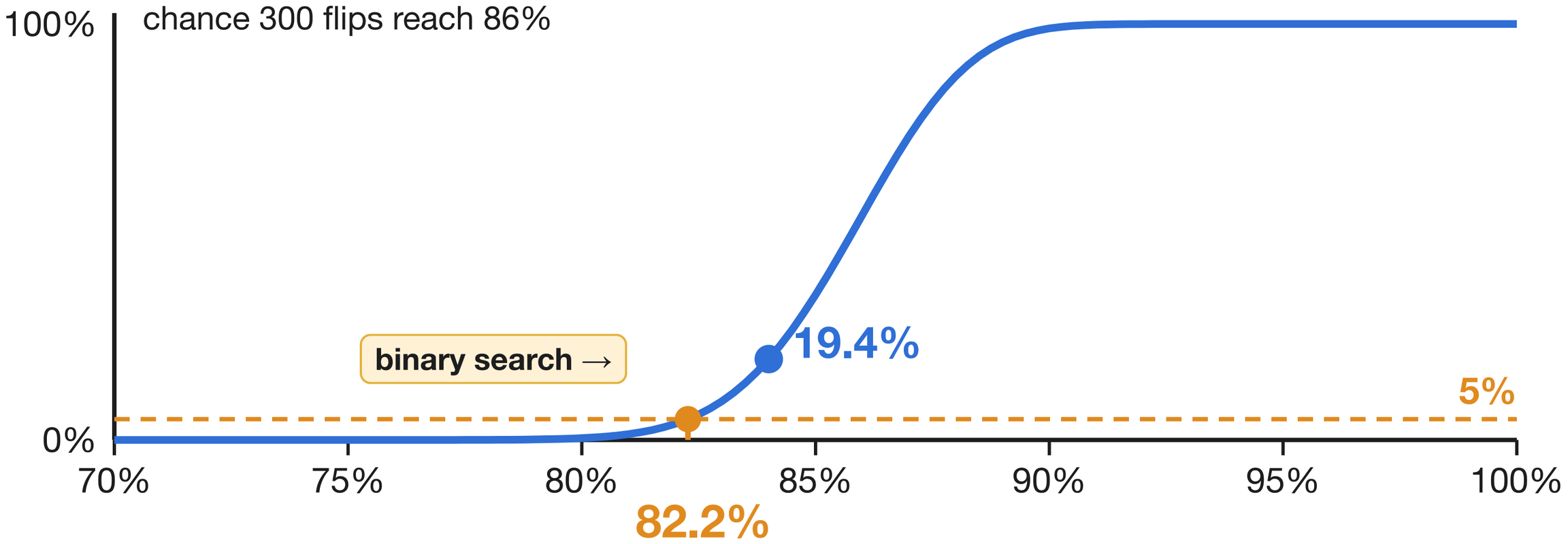


one simulated eval: 247 of 300

Unfortunately, yes, and too often:

19.4% of a million simulated evals do.

Below what $A(\pi)$ would 300 flips **rarely reach 86%**, by chance?



With chance at least 95%, $A(\pi) \geq \text{Floor}(\text{its rewards},\ 5\%) = 82.2\%$

Floor, for rewards of 0 or 1: the one-sided Clopper-Pearson limit, 1934.

`Floor(scores: list[float], c: float)`

takes	one score per request: a reward (0 to 1) or a difference (–1 to 1), and c, the chance we allow it to be wrong (5% so far)
returns	a floor F with $\Pr\left[\mathbb{E}[\text{score}] < F\right] \leq c^*$ * So across claims, these chances add up.

Deploying when Floor clears our bar

no longer assumes

- A ~~enough tries looked good~~
- B ~~I tried the right requests~~

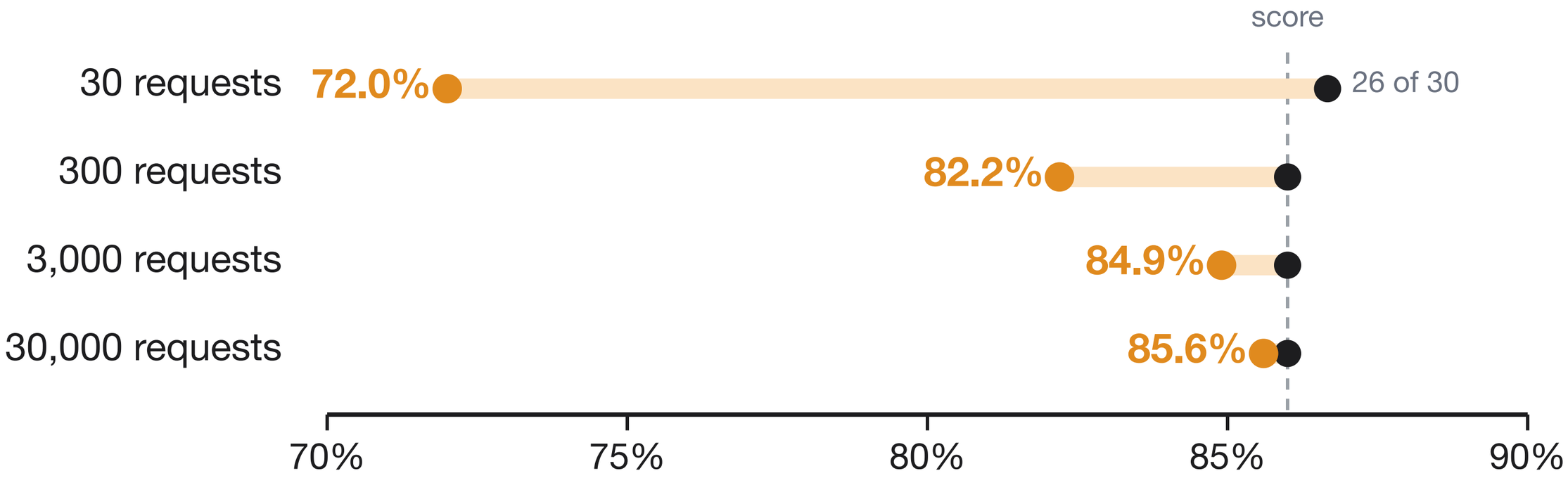
but assumes

- D **requests sampled at random**
- C next month is like this month

and that nothing was picked by looking at these scores

A lower confidence bound. Rewards of 0 or 1: Clopper and Pearson, 1934. Other numbers: Waudby-Smith and Ramdas, 2024.

What does 86% guarantee with 30 requests? With 3,000?



With ten times the data points, the gap is divided by about 3: $\sqrt{10} \approx 3$.

$$\text{Floor} \approx S - \sqrt{\frac{2\,S(1-S)\,\ln(1/c)}{n}}$$

Floor, for rewards of 0 or 1: the one-sided Clopper-Pearson limit, 1934.

Does Floor's guarantee still hold?

1. Score three programs on these 300 requests. Report the best one's Floor.
   the program was picked by looking at these scores ✗

2. Score three programs on 100 of these requests.
   Report the best one's Floor on the other 200.
   the program was picked before seeing these 200 scores ✓

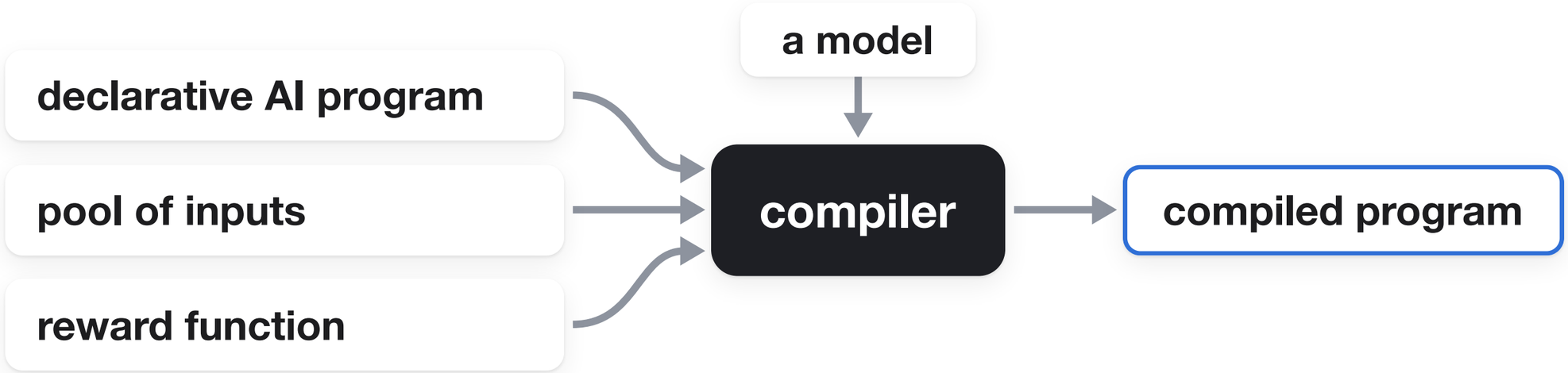
3. Drop 12 requests that failed oddly. Report Floor on the other 288.
   the requests were picked by looking at these scores ✗

4. Floor at 5% looks low. Report Floor at 10% instead.
   c was picked by looking at these scores ✗

Fix before seeing these scores: the program, the requests, and c.

Part 2

Learning from trial and error



Suppose it’s 2022.

resolve’s prompt, and what the model writes

```
Decide, then reply.
Message: Our 6:05 pm flight was cancelled. Next flight tonight?
Reply: Decide, then reply.
Message: My bag never arrived.
Reply: Decide, then reply.
Message:
```



What can we assume it does well?

A Can you move both of us to Friday?
trip A B

L Can I get a refund on my basic fare?
trip L basic fare

S Our 6:05 pm was cancelled. Tonight?
trip S K

R I'd like to talk to a person, please.
trip R

M Is my checked bag included?
trip M basic fare

understand

Find who and what they want.

Message: 6:05 pm cancelled. Tonight?
Trip: WX3N5Q, 2 on the trip
Who: 2 travelers
Wants: rebook tonight

the airline's systems

```
search_flights(tonight)
→ 8:40 pm, 3 seats

policy_for(rebook)
→ free change
```

resolve

Decide, then reply.

Facts: 8:40 pm, 3 seats
Action: rebook 2 on 8:40 pm
Reply: You're both on the 8:40 pm. Seats 14C, D.

We just got one right. What could we do with it?

A Can you move both of us to Friday? ✗
trip A B

L Can I get a refund on my basic fare? ✗
trip L basic fare

S Our 6:05 pm was cancelled. Tonight? ✓
trip S K

R I'd like to talk to a person, please. ✓
trip R

M Is my checked bag included? ✗
trip M basic fare

understand

Find who and what they want.

```
6:05 pm cancelled · 2 on the trip
→ 2 travelers, rebook tonight
```

```
A person, please · 1 on the trip
→ 1 traveler, a person
```

```
Message: Is a checked bag included?
Trip:    MB6Y9K, basic fare
Who:     1 traveler
Wants:   bag question
```

the airline's systems

```
policy_for(bags)
→ basic: carry-on only
```

resolve

Decide, then reply.

```
Facts: 8:40 pm, 3 seats
→ rebook 2 on 8:40 pm
```

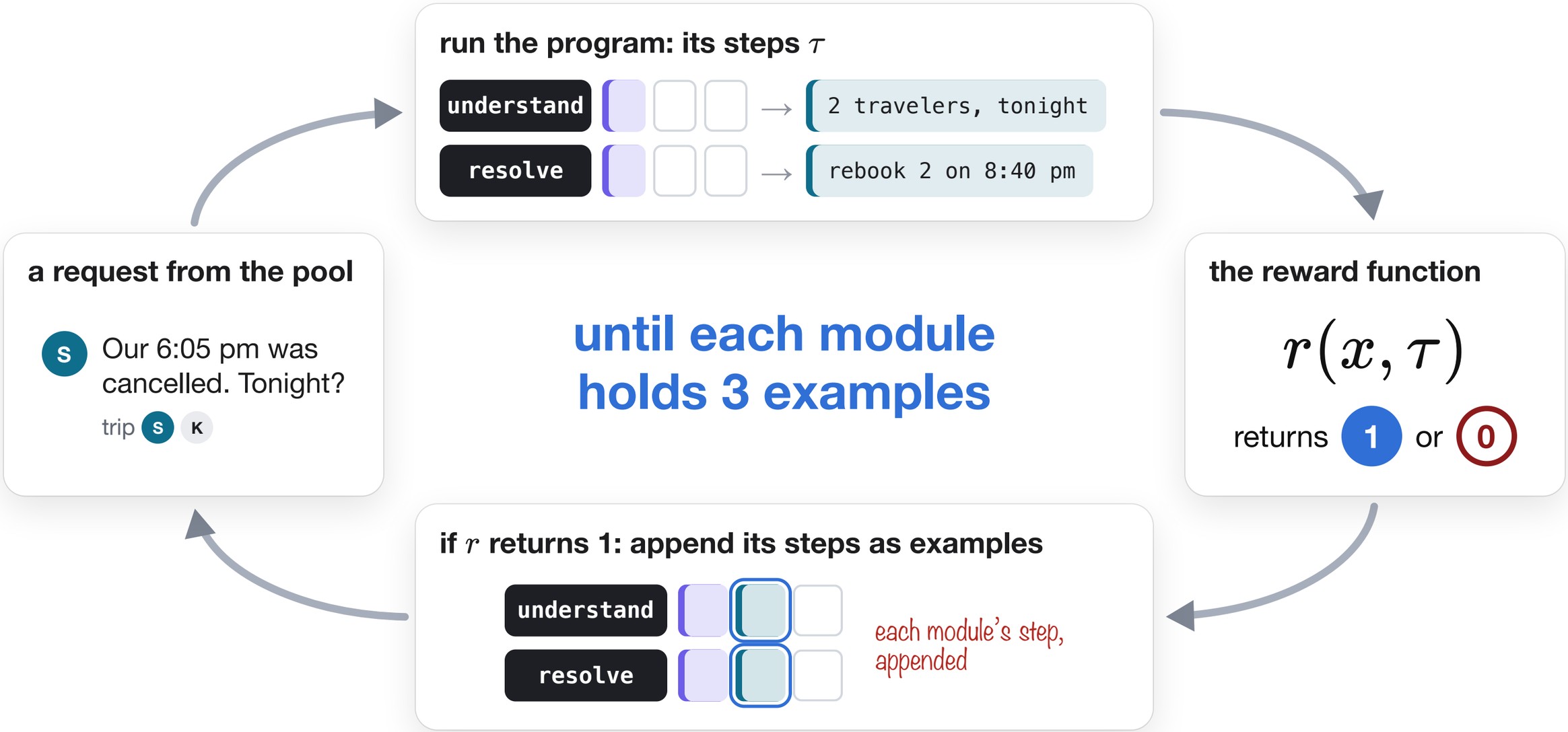
```
Facts: always connect
→ escalate
```

```
Facts:  basic: carry-on only
Action: answer
Reply:  Yes, one checked bag
        is included.
```

✗

That was the last request. What next?

DSP, 2022. DSPy, 2023. Many-shot: Agarwal et al., 2024. As a gradient: REINFORCE, Williams, 1992.



The context window is full. But do we know if we produced "the best" three examples?

BootstrapFewShot: DSP, 2022. DSPy, 2023.

the order we meet requests

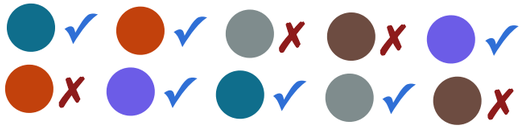


how many examples



a new random seed:

different rollouts, in a different order



how many attempts:

try each request 3 times, keep a rare success



keep the best: random search

	on 300 other requests	examples it kept, in order		
original program:	76%			
candidate program A:	83%	6:05 pm	a person	a bag
candidate program B:	88%	6:05 pm	a person	**both of us**
candidate program C:	84%	a bag	6:05 pm	**both of us**

kept after 6:05 pm, a person

```
understand
Both of us to Friday · 2 on the trip
→ 2 travelers, move to Friday
```

```
resolve
Facts: Fri 9:10 am, free change
→ move both to 9:10
```

kept after a bag, 6:05 pm

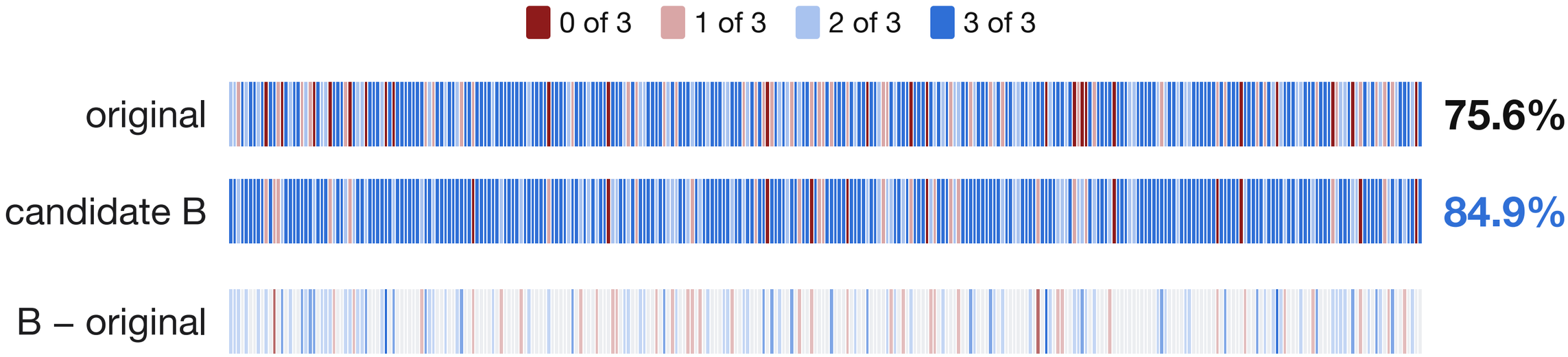
```
understand
Both of us to Friday · 2 on the trip
→ 2 travelers, which flight?
```

```
resolve
Facts: Fri 9:10 am, 2:45 pm
→ ask: 9:10 or 2:45?
```

Candidate program B looks best: 88%! But does it actually beat the original program?

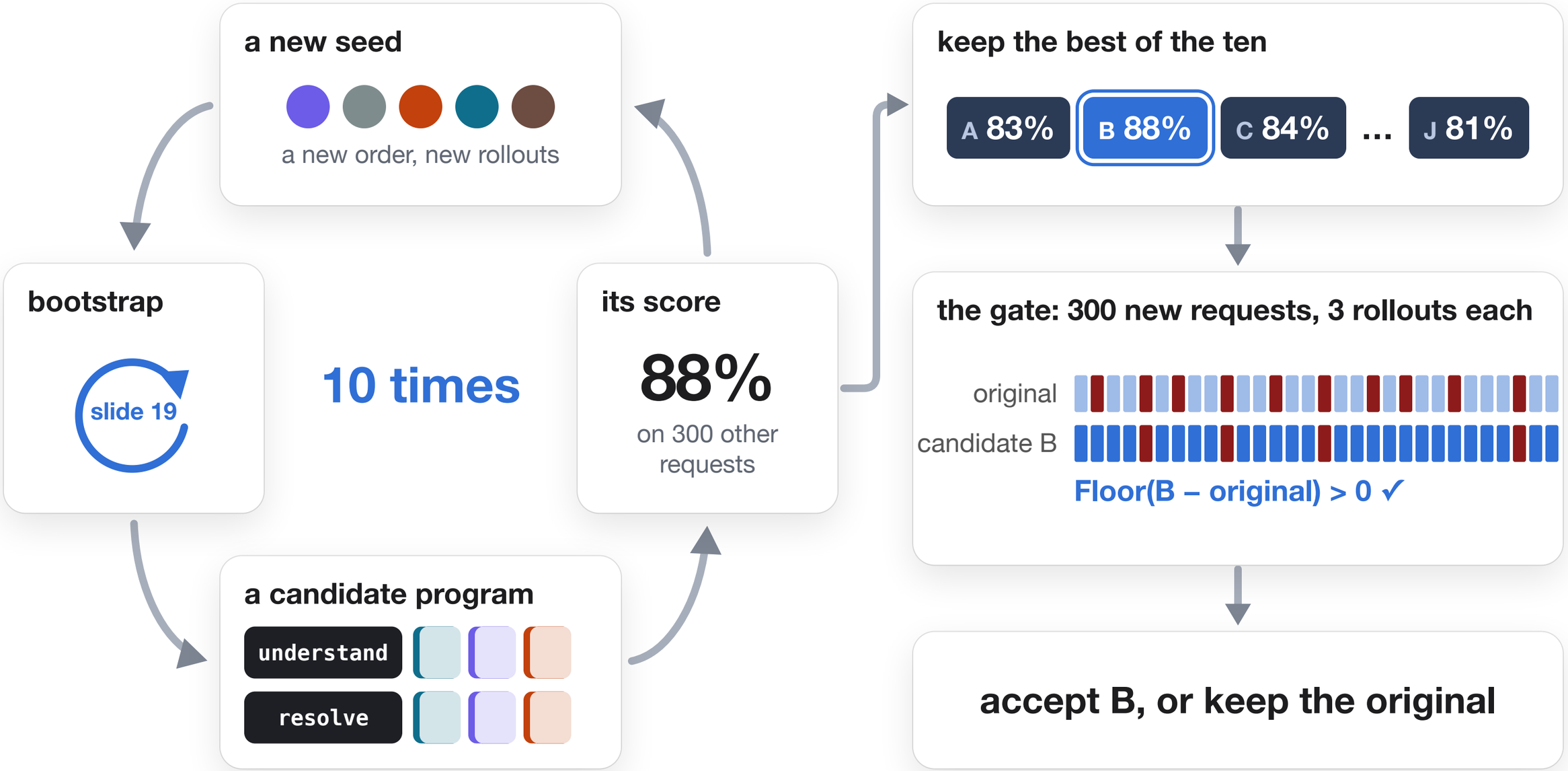
BootstrapFewShotWithRandomSearch: DSPy, 2023. Constructed examples.

300 new requests. Run each program 3 times on each, average per request, subtract:



$$\text{Floor}(\text{the differences},\ 5\%) = +5.2 \text{ points}$$

Candidate program B passes, with chance at least 95%.



BootstrapFewShotWithRandomSearch: DSPy, 2023. The gate is ours.

Part 3

Learning from search

What is random search throwing away?

how it spends rollouts

all 300 requests on a candidate that looks bad after 30 requests



7 of 30 pass

270 more requests, run anyway

what it remembers

nothing: each new candidate ignores every earlier score

~~83%~~ ~~88%~~ ~~84%~~ → ?

what it can change

only the examples, never the instruction

instruction: fixed

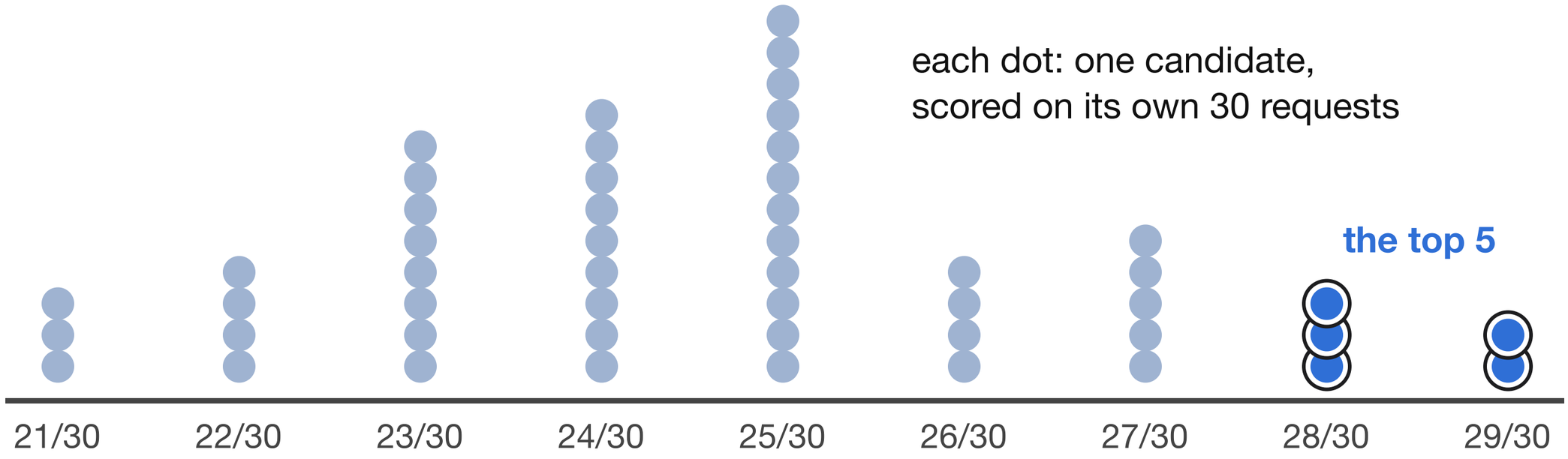
random search: 10 candidates × 300 requests = 3,000 rollouts



50 candidates × 30 requests

the top 5 × 300





Each candidate got its own 30 requests. What's unfair about that?

the original program: scored once on every training request

30 easy requests

candidate X **28/30**



original **27/30**



X: +1 request

30 hard requests

candidate Y **23/30**



original **18/30**



Y: +5 requests

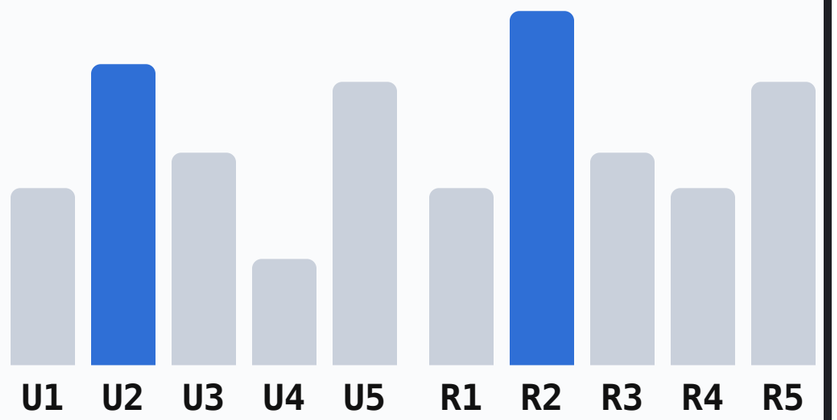
Random search does not learn over time from its exploration. Could we do better?

every try so far

U1	R1	83%
U2	R2	88%
U3	R3	84%
U4	R4	81%
U5	R5	86%
U2	R4	85%

→

a model of which pieces did well



→

the next try

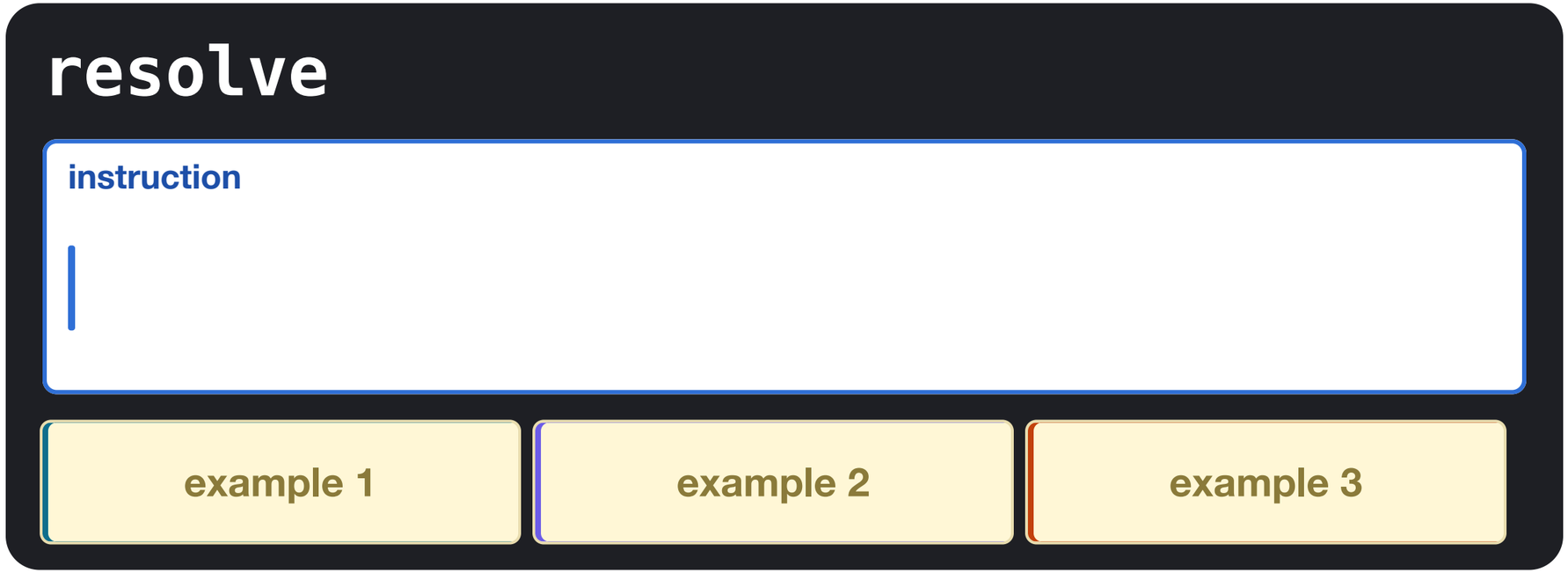
U2 R5

assumes a piece that
did well with some partners
does well with others

So far, the pieces differ only in their examples. What else should they vary in?

Bayesian optimization: TPE, Bergstra et al., 2011.

Suppose it's late 2023.



What information do you need to see to write better instructions for the `resolve` module?

GPT-4: OpenAI, March 2023.

our program

a summary of the requests

runs that worked

earlier instructions, scored

a random tip

~~failures, with why~~ ✗

2025

→ **proposer LM** →

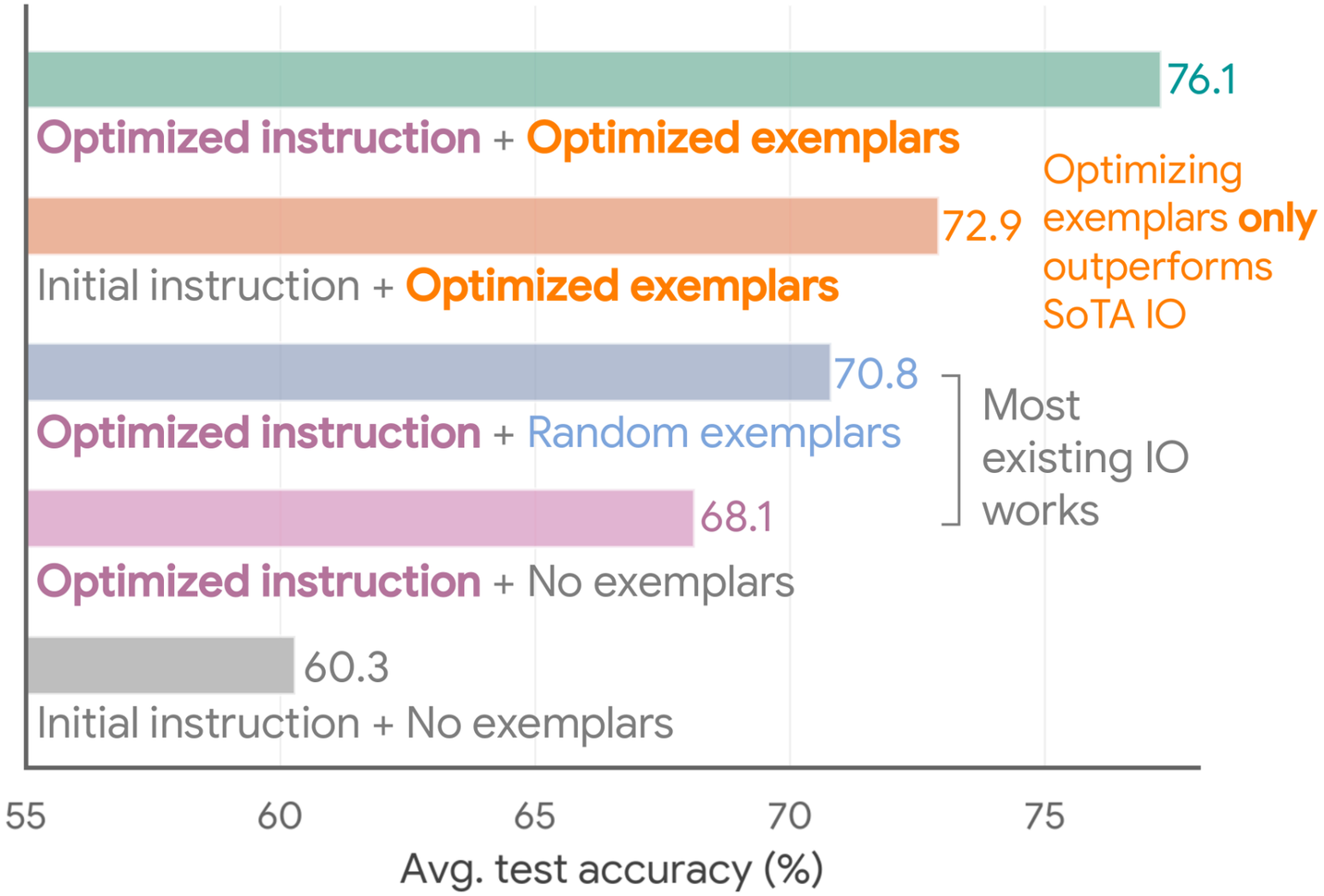
A Rebook everyone on the trip, unless they name who is moving.

B Connect them to a person as soon as they ask. Don’t try to solve it first.

C Confirm the new flight and seat in one short message.

What can an instruction say that examples can’t? And the reverse?

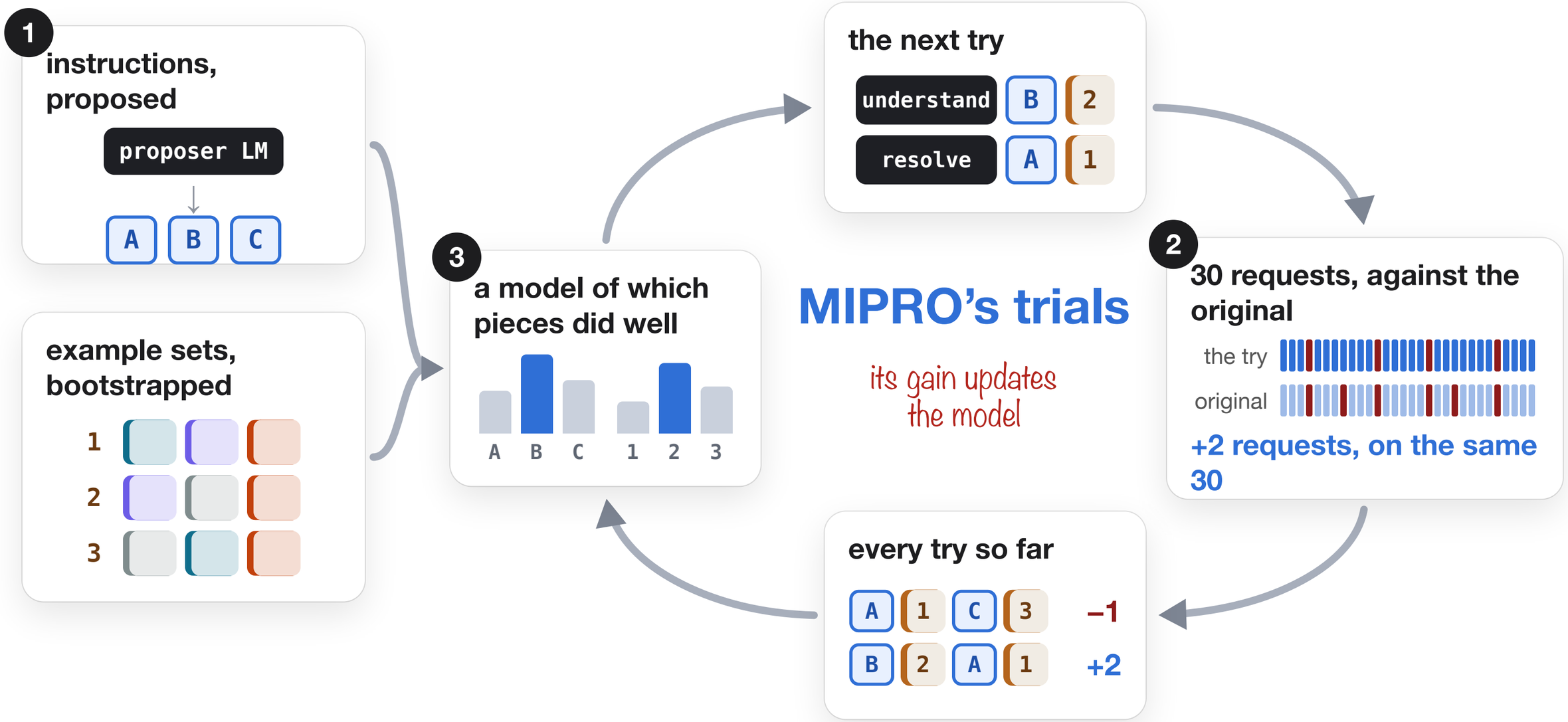
Grounded proposal: Opsahl-Ong et al., EMNLP 2024. Proposals constructed.



So, what's a reasonable overall algorithm with a late 2023 model?

Let's consider **MIPRO**: **M**ulti-prompt **I**nstruction **PR**oposal **O**ptimizer

Wan, Sun, Nakhost and Arik, "Teach Better or Show Smarter?," NeurIPS 2024, Figure 1 (PaLM 2, more than 20 tasks).



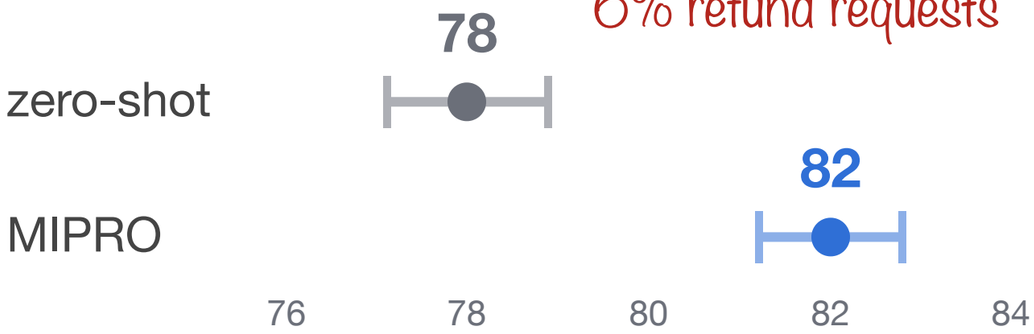
Three changes to random search. Are they worth it?

MIPRO: Opsahl-Ong et al., EMNLP 2024. The baseline on each minibatch is ours.

Team A ran MIPRO with an LLM judge

10,000 conversations, sampled at random

6% refund requests



Team B K by hand, judged by Kate

50 edge cases, picked by Kate

40% refund requests



D requests sampled at random

E the judge agrees with Kate

passes 12 conversations that Kate failed

B we picked the right requests, with Kate

judged by the target itself

Which assistant would you deploy today?

{ } Being declarative

does not replace

Being careful



Automated evals and statistics

does not replace

Looking at data

r(·) An executable reward function

does not replace

Dogfooding

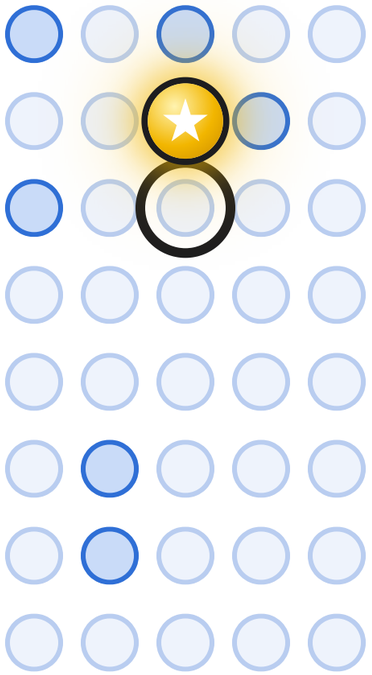
Before investing a lot of time building and evaluating MIPRO, is there a way to know whether it's worth building, and what assumptions it makes about the world?

Let's simulate both methods, in a world we write down. Four steps:

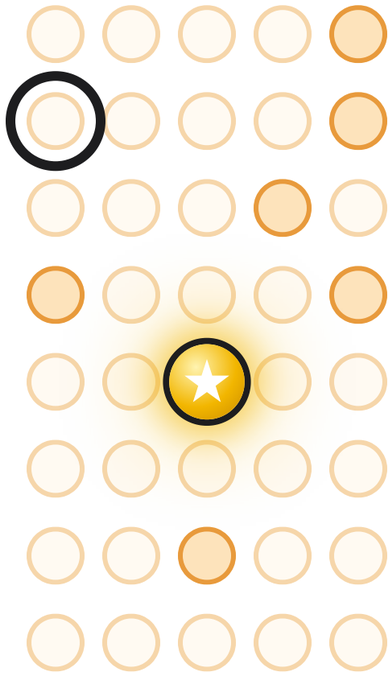
1. Write the world down: each piece's options, and how good each one is.
2. Build a simulator, and check it against data we have, e.g. HW1's rollouts.
3. Run random search and MIPRO in it, with the same rollouts.
4. Change one assumption at a time: where does MIPRO's lead survive?

1 Write the world down

`understand`

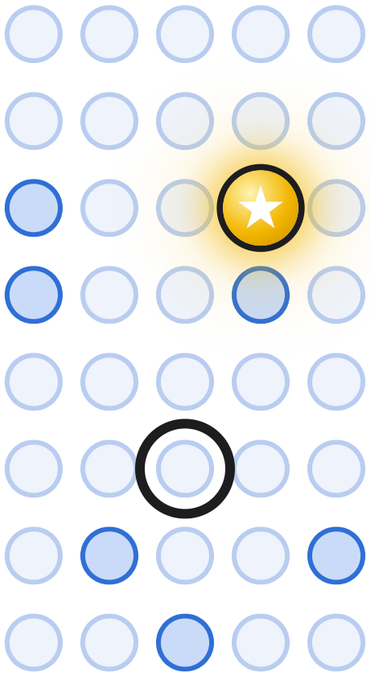


instruction

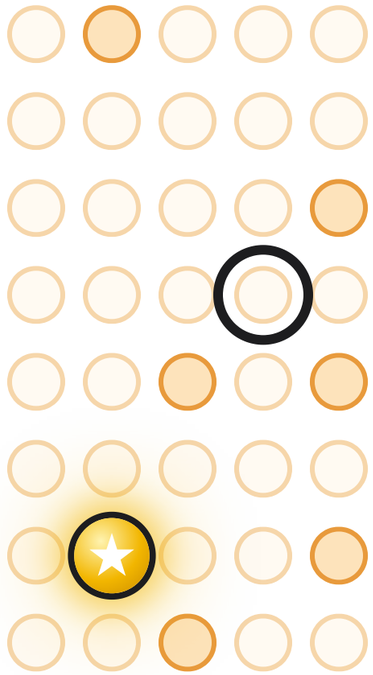


examples

`resolve`



instruction



examples



one option for a piece:
an instruction, or examples



a little better
than most



a star: fixes a whole
kind of failure

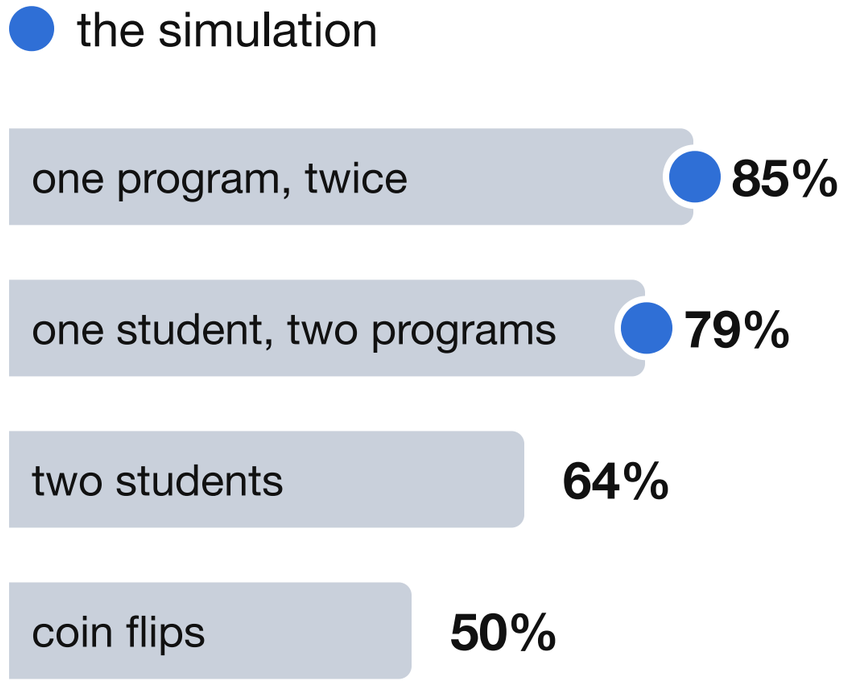


a program:
one option per piece

2 Build a simulator, and check it against data we have

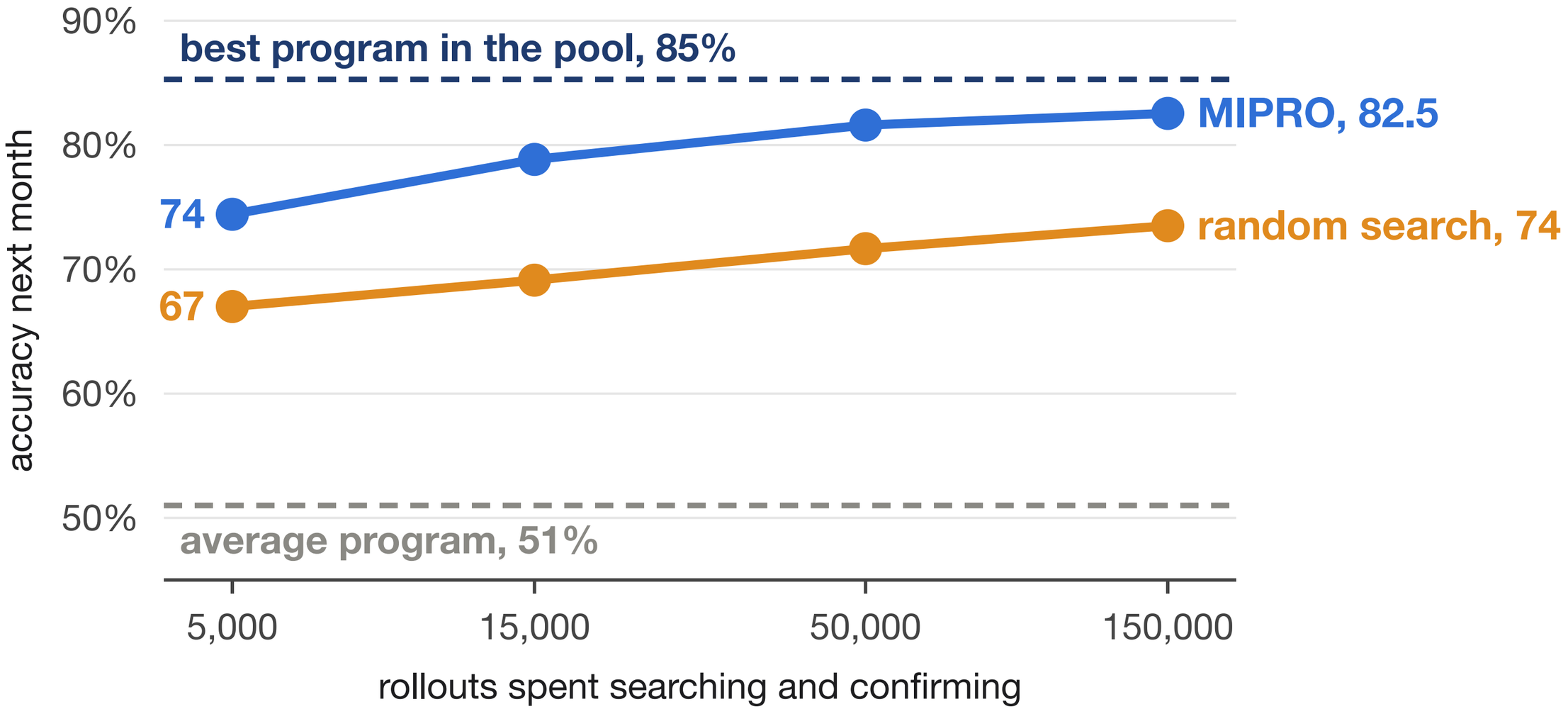
```
> Simulate a program of 4 pieces, 40 options
  each: a few stars, or a crowd.
  1. A rollout passes with a chance set by its
     options, the request, and noise.
  2. Fit the noise to HW1: one program run twice
     agrees on 85% of requests, two programs of
     one student on 79%.
  3. Give random search and MIPRO the same
     rollouts. Score what each returns on next
     month's requests.
  4. Tune each on 30 worlds. Report on 100 others.
```

HW1: how often two runs agree



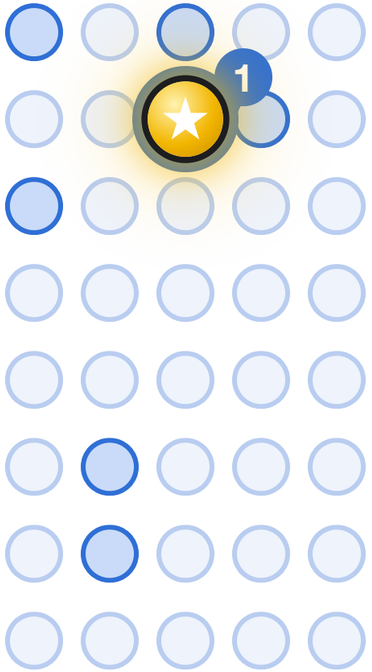
How close does random search get?

3 Run both methods



Why can’t random search catch up?

understand



instruction



examples

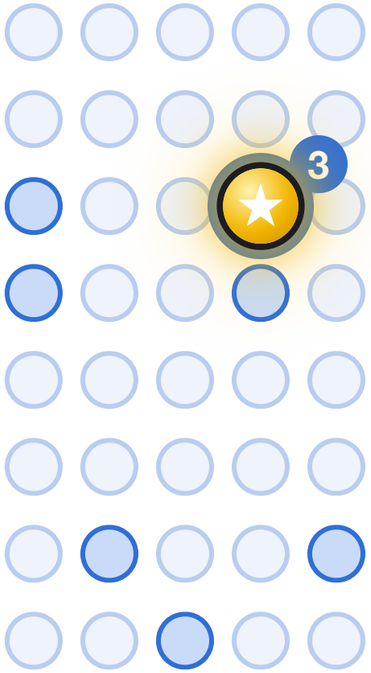
a piece's star

1 in 40

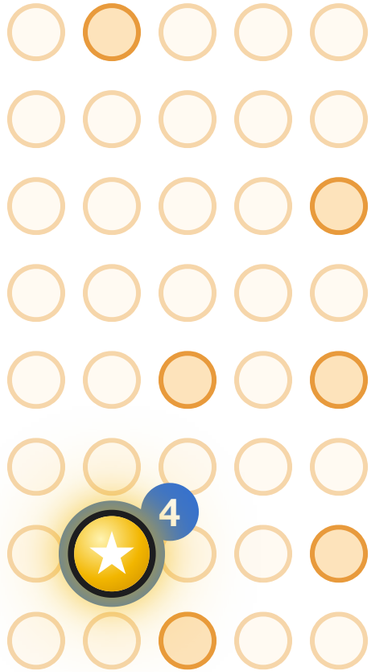
exactly two

1 in 280

resolve



instruction



examples

exactly three

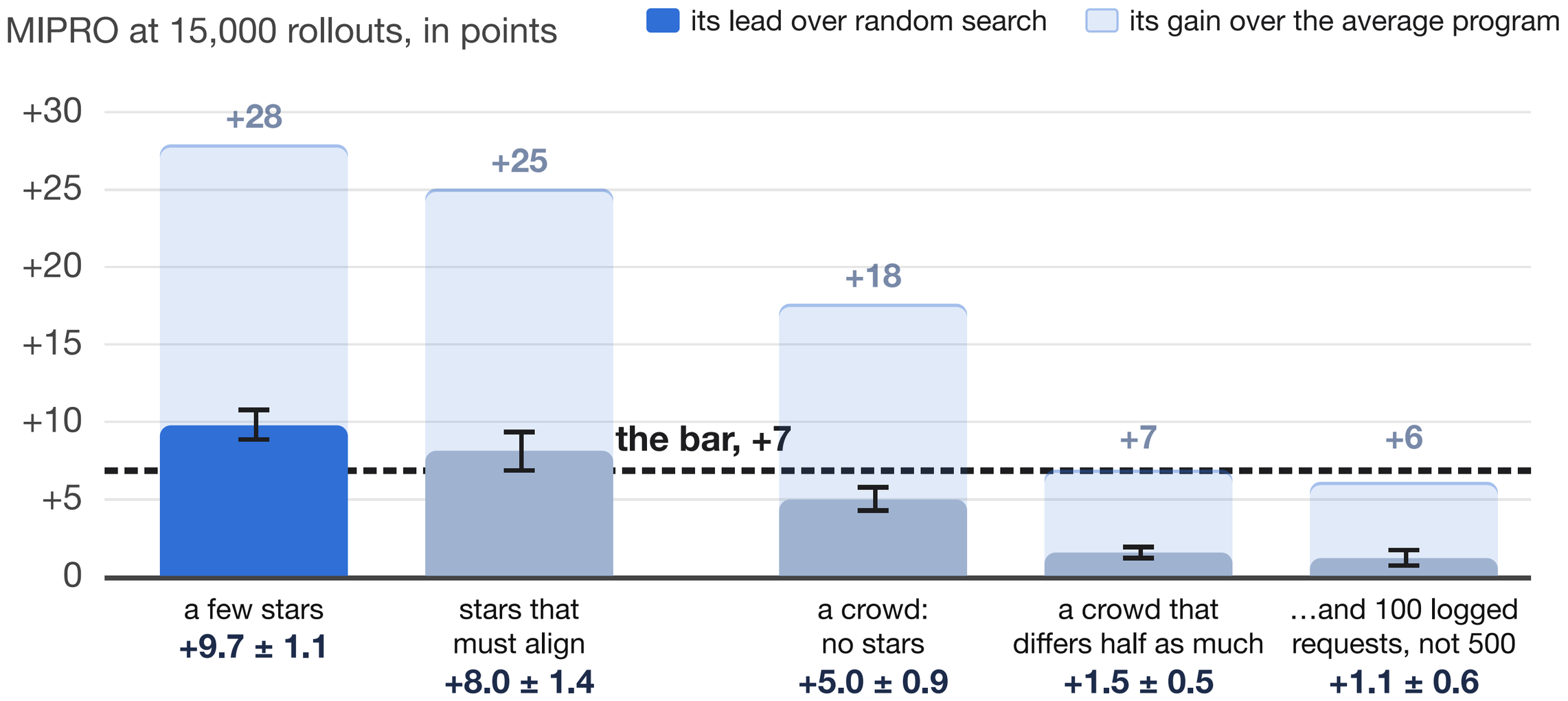
1 in 16,000

all four

1 in 2.6 million

What if there are no stars?

4 Change one assumption at a time



What would put stars in the pool?

Part 4

Learning from reflection

What else could we assume about the model?

Suppose it's 2025.

1

Can I get a refund on my basic fare?

```
understand  Wants:   a refund
resolve     Action:  refund $189
            Reply:   Your refund is on its way.
```

r = 0 ✗

2

`sys` **the airline's systems**

`issue_refund(KP4R8M) → refused: basic fare`

3

`{ }` **our reward function, opened**

- ✓ answers what they asked
- ✓ kind
- ✓ reply matches the action
- ✗ no forbidden refund
- ✗ reads policy_for first

4

J **a judge with Kate's note from last month**

K Basic fares can't be refunded. Offer a credit.

privileged: the program never sees it

feedback: "It promised a refund. Offer a credit instead."

Can I get a refund on my basic fare? ✗

Our 6:05 pm was cancelled. Tonight? ✓

I'd like to talk to a person, please. ✓

← a minibatch of 3 requests

understand Wants: `a refund`

resolve Action: `refund $189`

Reply: `Your refund is on its way.`

sys issue_refund → refused: basic fare

{ } ✗ no forbidden refund
✗ reads policy_for first

J offer a credit instead

reflect

resolve

Decide, then reply.

```
+ Read policy_for
before a refund.
```

```
+ Basic fares
can't be refunded:
offer a credit.
```

Is the rewrite better?

the Pareto front: the versions that are best on some request

6 held-out requests

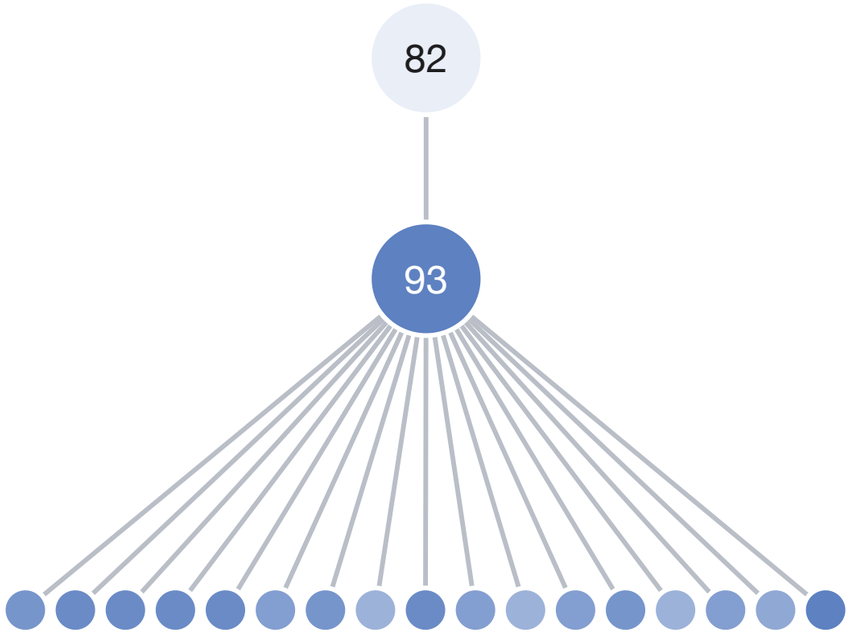
	three of us	refund	rebook	a person	a bag	change fee	
P0	✗	✗	✓	✓	✓	✗	3 of 6
P1	✗	✓	✓	✓	✗	✓	4 of 6
	no one						

draw the next parent

P0 best on 3 requests | **P1** best on 4 requests

Why not just always improve the best version, P1?

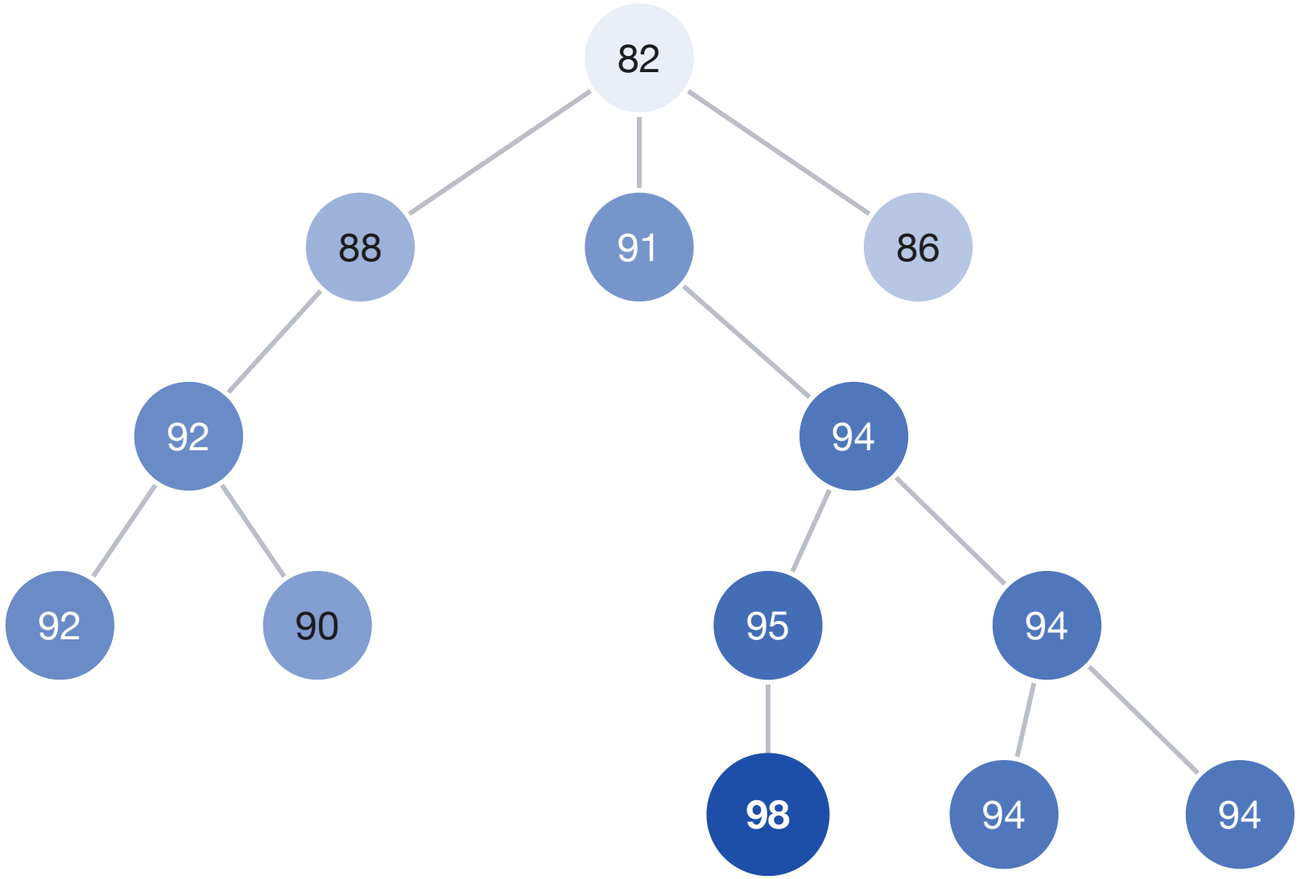
always improve the best



17 tries on one parent, none above 93

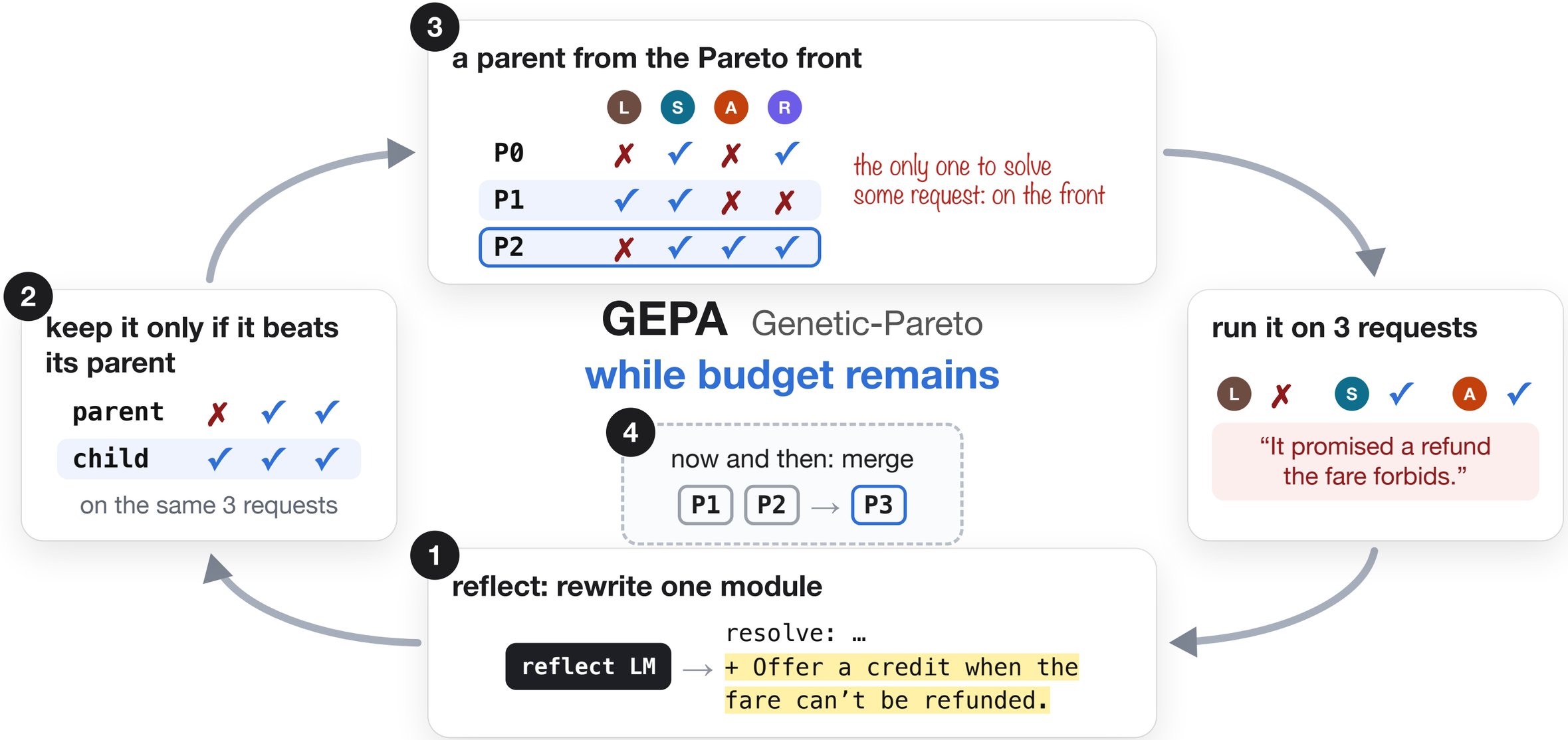
best found: **93** points

draw from the bests, then merge



best found: **98** points

GEPA: Agrawal et al., ICLR 2026, Figures 25a and 5 (PUPA, GPT-4.1 Mini), redrawn, scores rounded.



What did the rewrites actually add?

GEPA: Agrawal et al., ICLR 2026.

Our GEPA run read the refund failures in our 300 requests and rewrote `resolve` 50 times.

two lines it added

```
+ Basic fares can’t be refunded: offer a credit.
```

a rule: holds next month

```
+ If the booking is KP4R8M, offer a $50 credit.
```

one booking: never seen again

The best rewrite scores 86% on the same 300 requests.

Wouldn’t rewrites written from its own errors just overfit?

Constructed example.

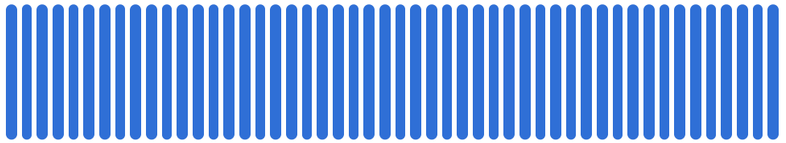
How do we share our 5% chance of being wrong?

One rewrite,
fixed before we sampled the 300



$A \geq$ **82.2%**

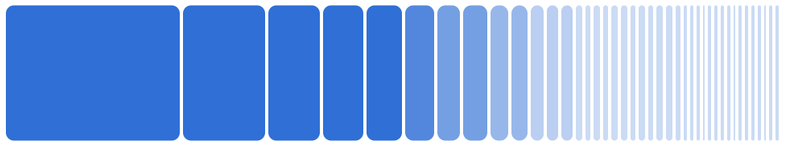
Fifty rewrites,
written before we sampled the 300



best: $A \geq$ **78.8%**

Not GEPA's fifty: it wrote them by reading these 300.

Any texts, GEPA's included,
with shares fixed before
we sampled the 300



5% × *P*(*π*)

Before we sample the 300, which texts would you bet on?

Adding up the chances of being wrong: the union bound.

1 Fix a prior $P(\pi)$. It may read anything but the requests of step 2.

2 Sample n requests **D** independently at random from $\mathcal{D}$.

3 Then, with chance at least 95%, for every π at once:

$$A(\pi) \;\geq\; \mathrm{Floor}\big(\text{its rewards},\; 5\% \times P(\pi)\big)$$

Occam's Razor bound

a plain instruction, likely given the signature, pays little

Langford, "Tutorial on Practical Prediction Theory for Classification," JMLR 2005, Theorem 4.1.

All three score 86%. A language model can place the bet.

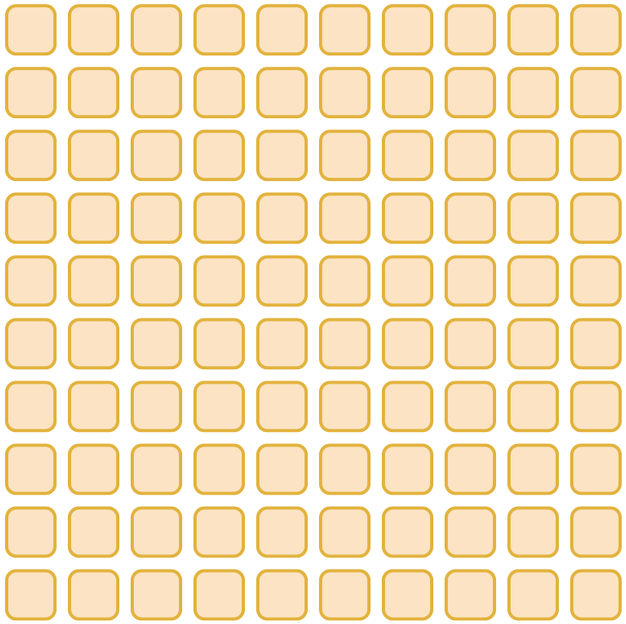
	$P(\pi)$, alone	$P(\pi)$, given the signature and logs	guarantee
`Before promising a refund, check policy_for.`	e^{-60}	$\boldsymbol{e^{-15}}$	$\boldsymbol{A \geq 72.4\%}$
`Before promising a refund, check policy_for. Offer a credit when the fare can't be refunded.`	e^{-102}	$\boldsymbol{e^{-28}}$	$\boldsymbol{A \geq 66.9\%}$
`Car @$32 blog — tissue || 324r 1 refund policy_for .. ;;`	e^{-163}	$\boldsymbol{e^{-159}}$	$\boldsymbol{A \geq 36.1\%}$

Anything but those 300: the signature, last month's logs.

Priors over prompts: Akinwande et al., ICLR 2024. Qwen3.5-397B, given the signature and three logged conversations. Scores constructed.

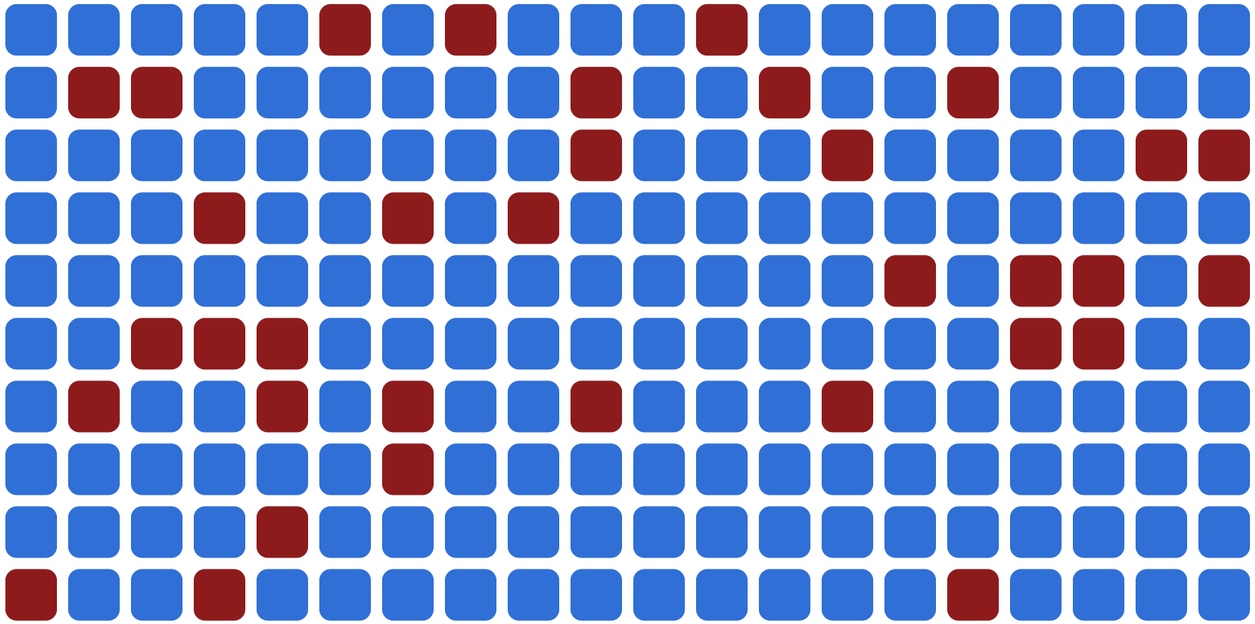
Let’s split our 300 requests.

GEPA reads 100



50 rewrites, three kept

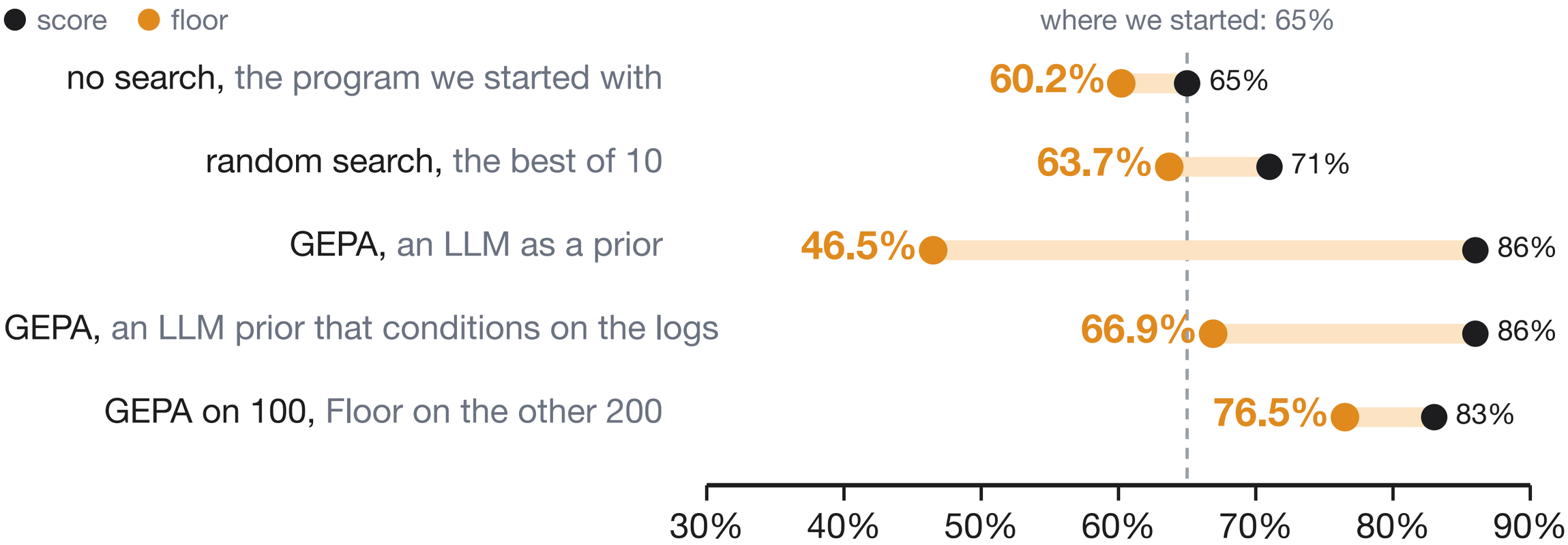
Floor reads the other 200



the best of the three: 166 of 200 pass, $A \geq$ **76.5%**

Fixed before we look at the 200: $P(\pi) = 1/3$ each.

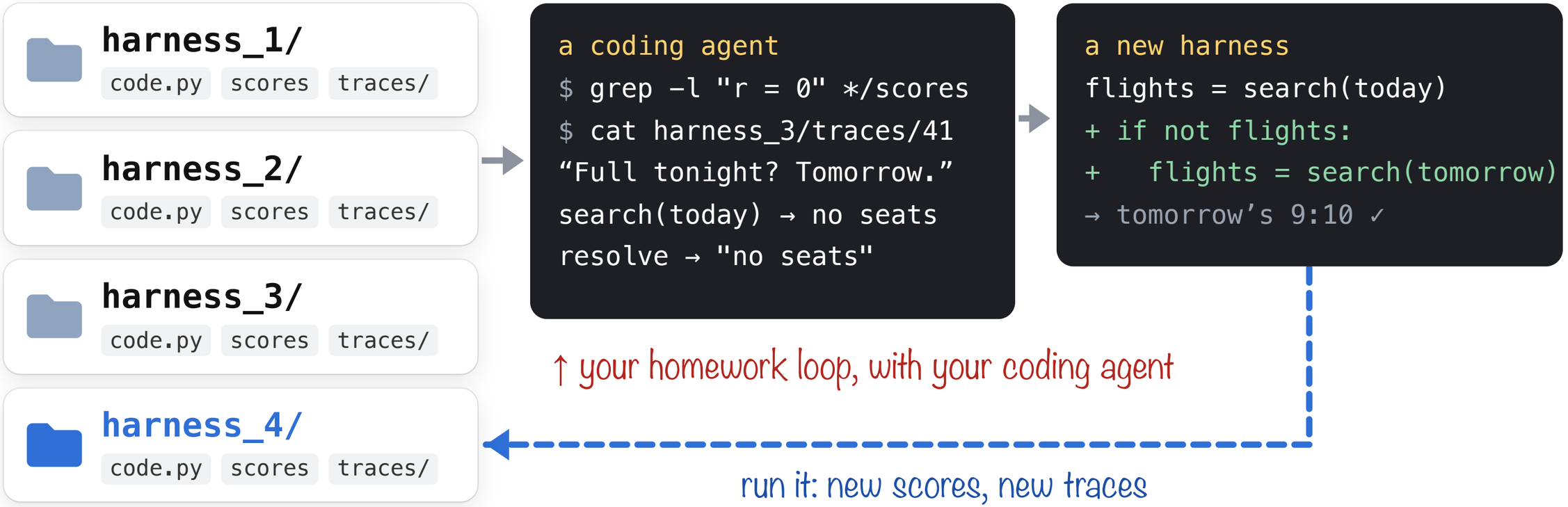
What does each search's score guarantee?



Search raises the score. The floor pays for how much the search could pick.
A gain is shown only where the floor clears 65%, where we started.

Scores constructed. Prior costs: Qwen3.5-397B, as on the bet slide.

2022 | 2023 | 2025 | **2026**



What should the agent read?

Meta-Harness: Lee et al., 2026.

median accuracy of the harnesses it writes, by what the agent reads



MIPRO's and GEPA's pieces, handed to a coding agent as data, tools and advice

`examples/` `minibatch.py` `scores.db` `pareto.py` `merge.py` `advice.md`

How much of an optimizer must stay built in?

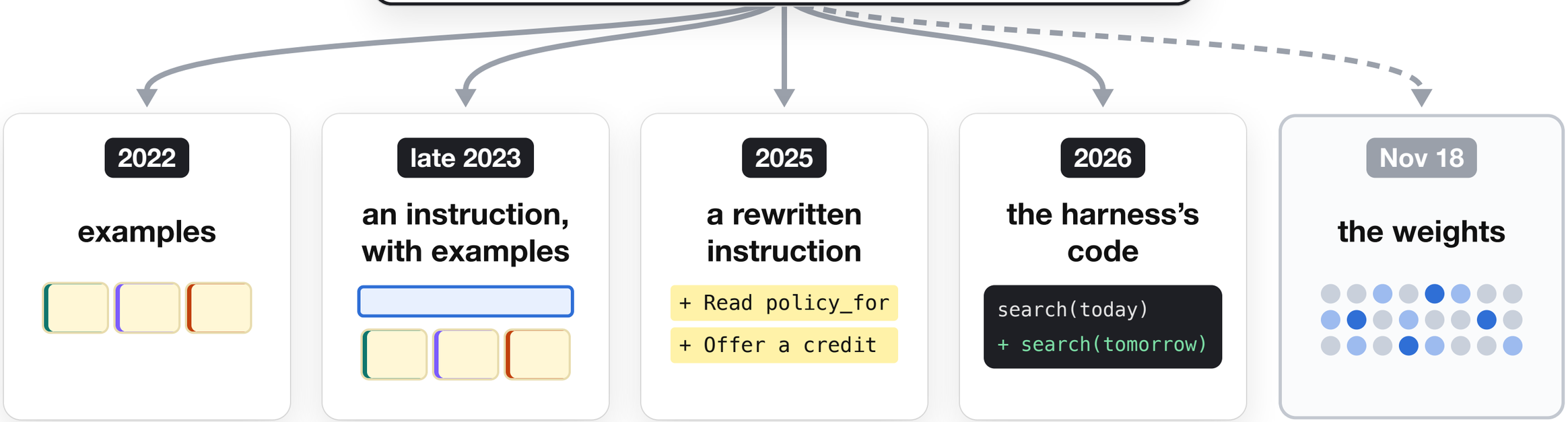
Meta-Harness: Lee et al., 2026, Table 3 (median over proposed harnesses).

`resolve`

reads message, need, flights, policy

writes action, reply

Never promise a refund the fare rules forbid.



Compilers change with what models can do.
A well-designed program outlasts them.

But what did all these algorithms leave us wanting?

$$\max_{\Theta} \quad \mathbb{E}_{x \sim \mathcal{D}} \quad \mathbb{E}_{\tau \sim \pi_{\Theta}(x)} \quad \left[r(x, \tau) \right]$$

Is every failure easy to describe in an instruction?

November 18

Is what we want an average over one distribution?

a forbidden refund must almost never happen, and only 6% ask for one

Can we always run new rollouts?

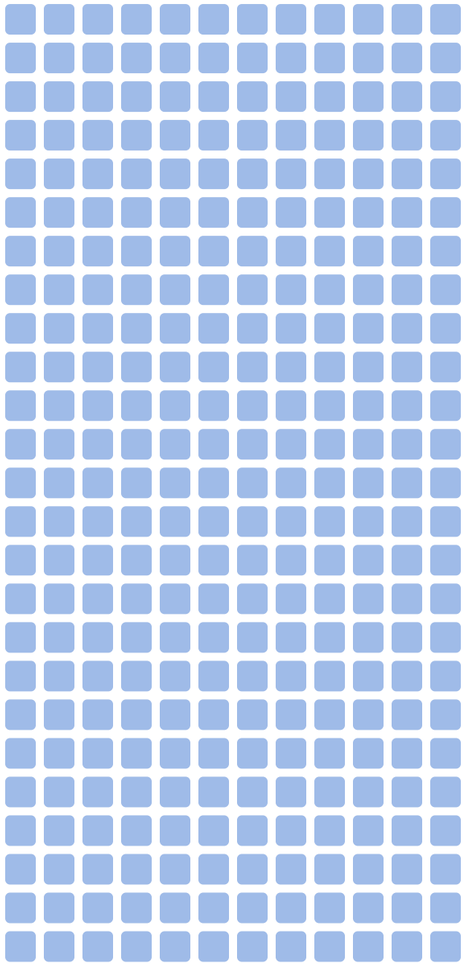
often there are only last month's logs

HW2

Is the reward what we want?

the judge passes 12 conversations that Kate failed

What should a compiler actually promise?



300 new conversations,

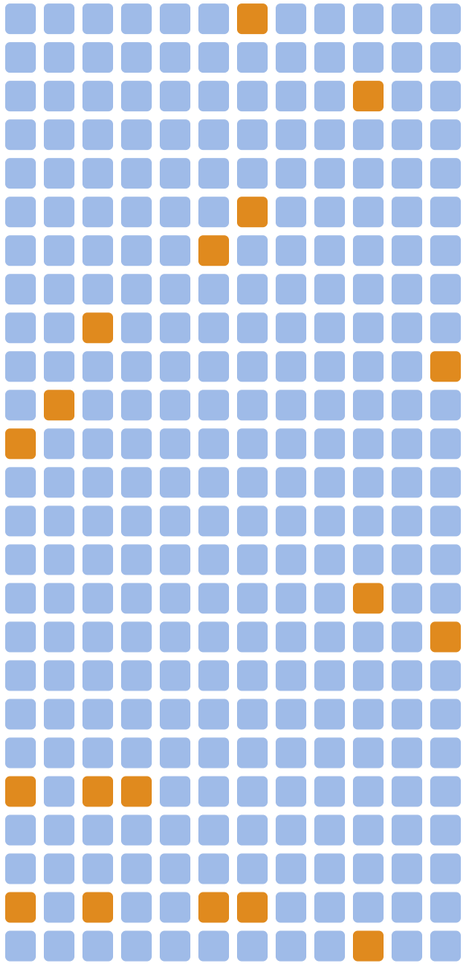
D sampled at random.

Kate checked each one: no forbidden refund!

$$\mathrm{Floor}(300\ \mathrm{clean},\ 5\%) = 99.0\%$$

Forbidden refunds: below 1 in 100,
with chance at least 95%.

30,000 clean checks for 1 in 10,000!
But what did those checks cover?



Only **18** asked for a refund.

$$\text{Floor}(18 \text{ clean}, \ 5\%) = 84.6\%$$

Per refund request, failures below **15.4%**.

Check refund requests on purpose:
300 clean, failures below **1%**.

November 18

When does an improvement belong in the **weights**?