

CS11-711 Advanced NLP

# Inference-Time Scaling

Sean Welleck

**Carnegie  
Mellon  
University**

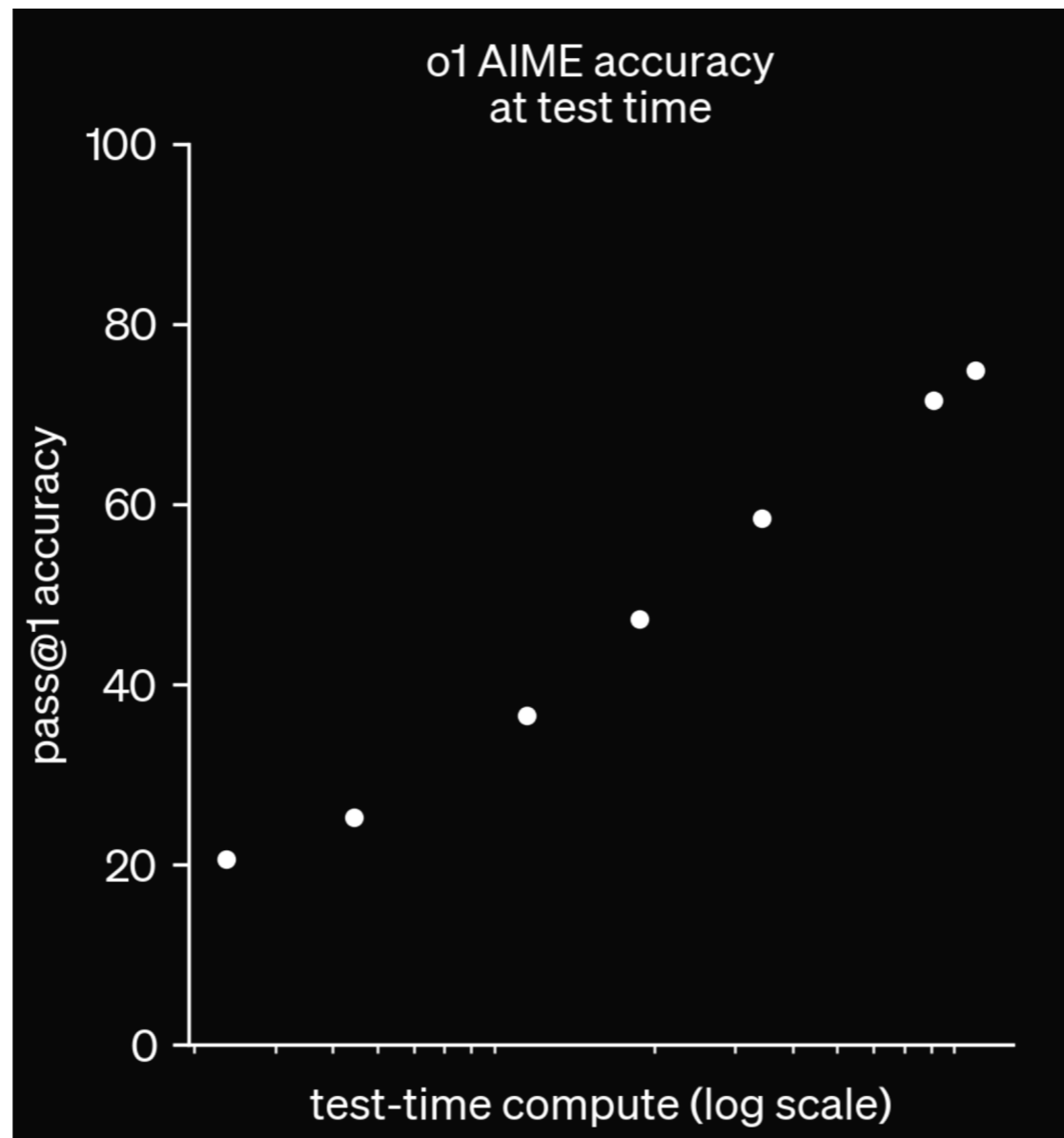


# Inference

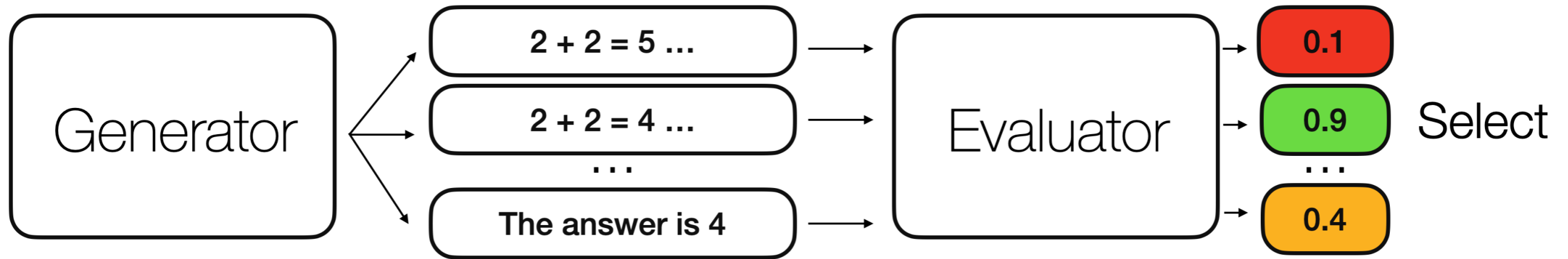
- Generate outputs with a model and an algorithm

# New dimension of scaling

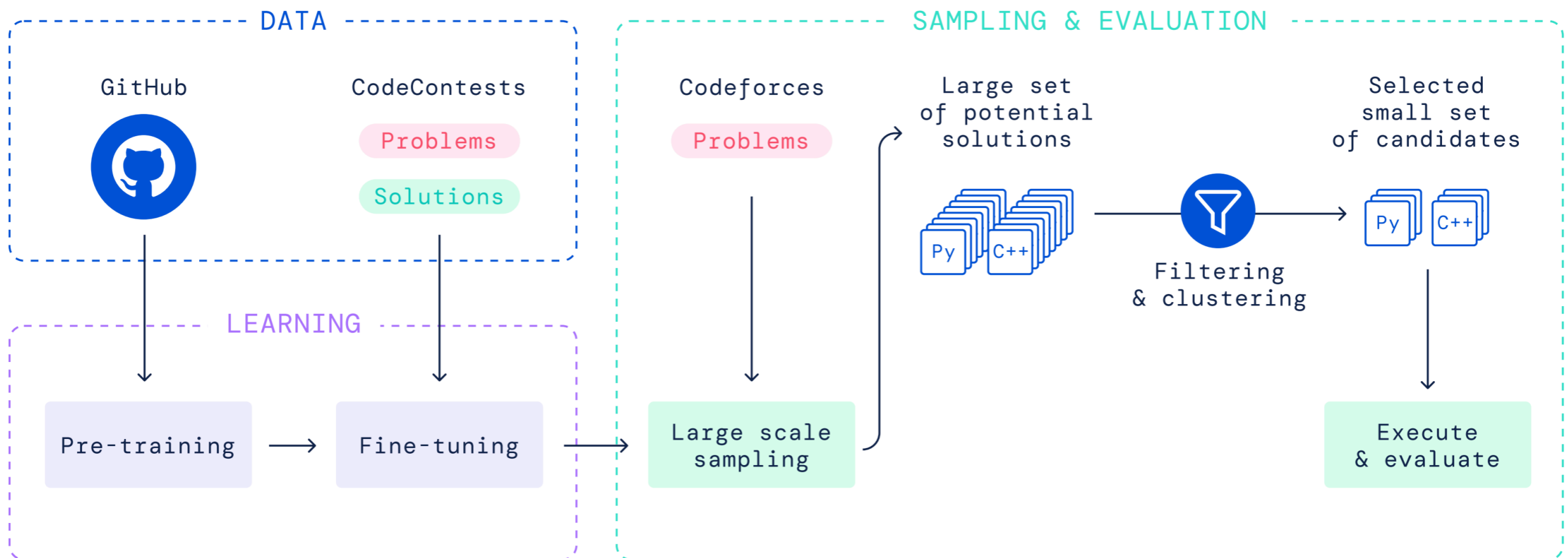
- “Test-time compute”



# Strategy 1: generate multiple times



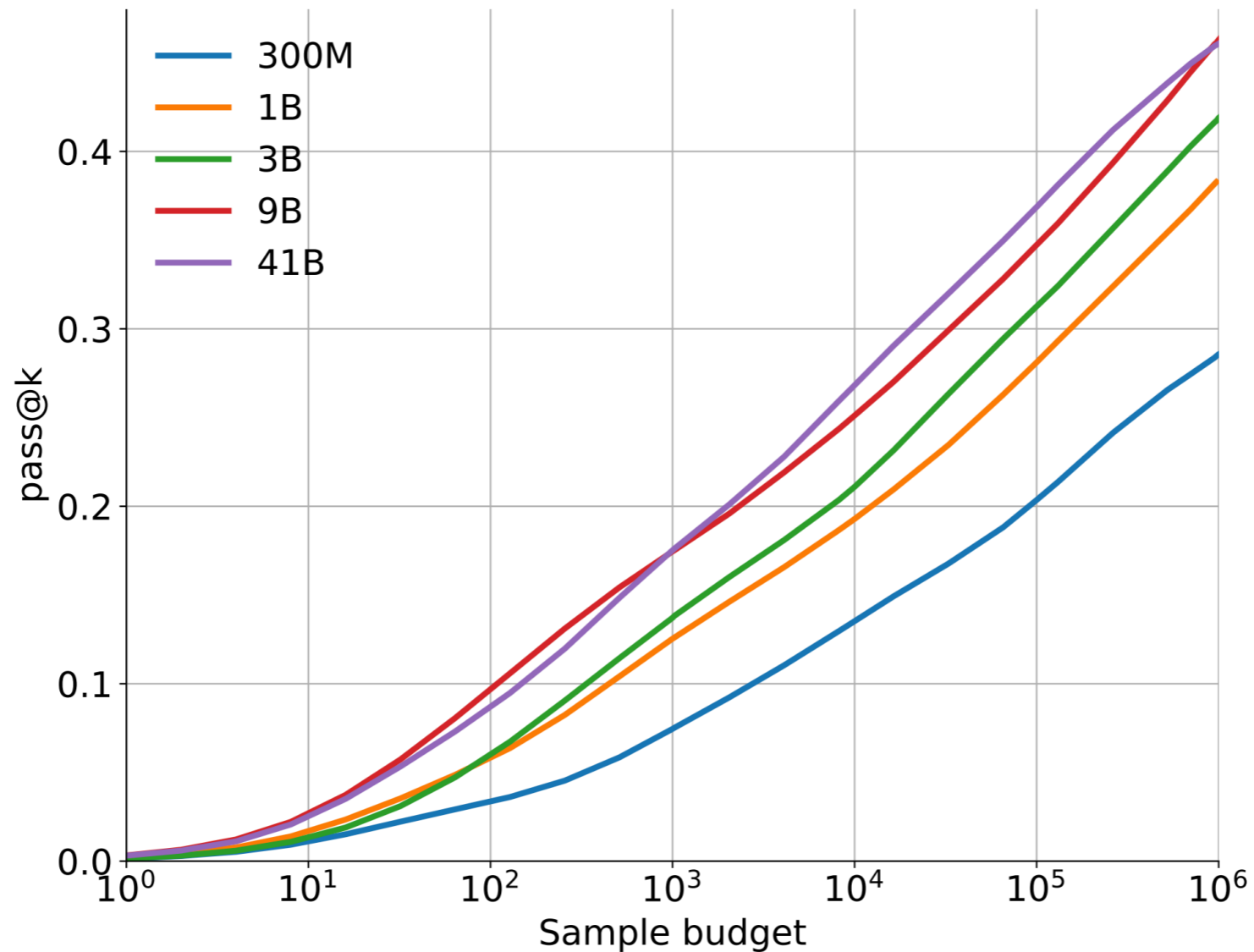
# Strategy 1: generate multiple times



## Overview of AlphaCode.

[Deepmind 2022]

# Strategy 1: generate multiple times

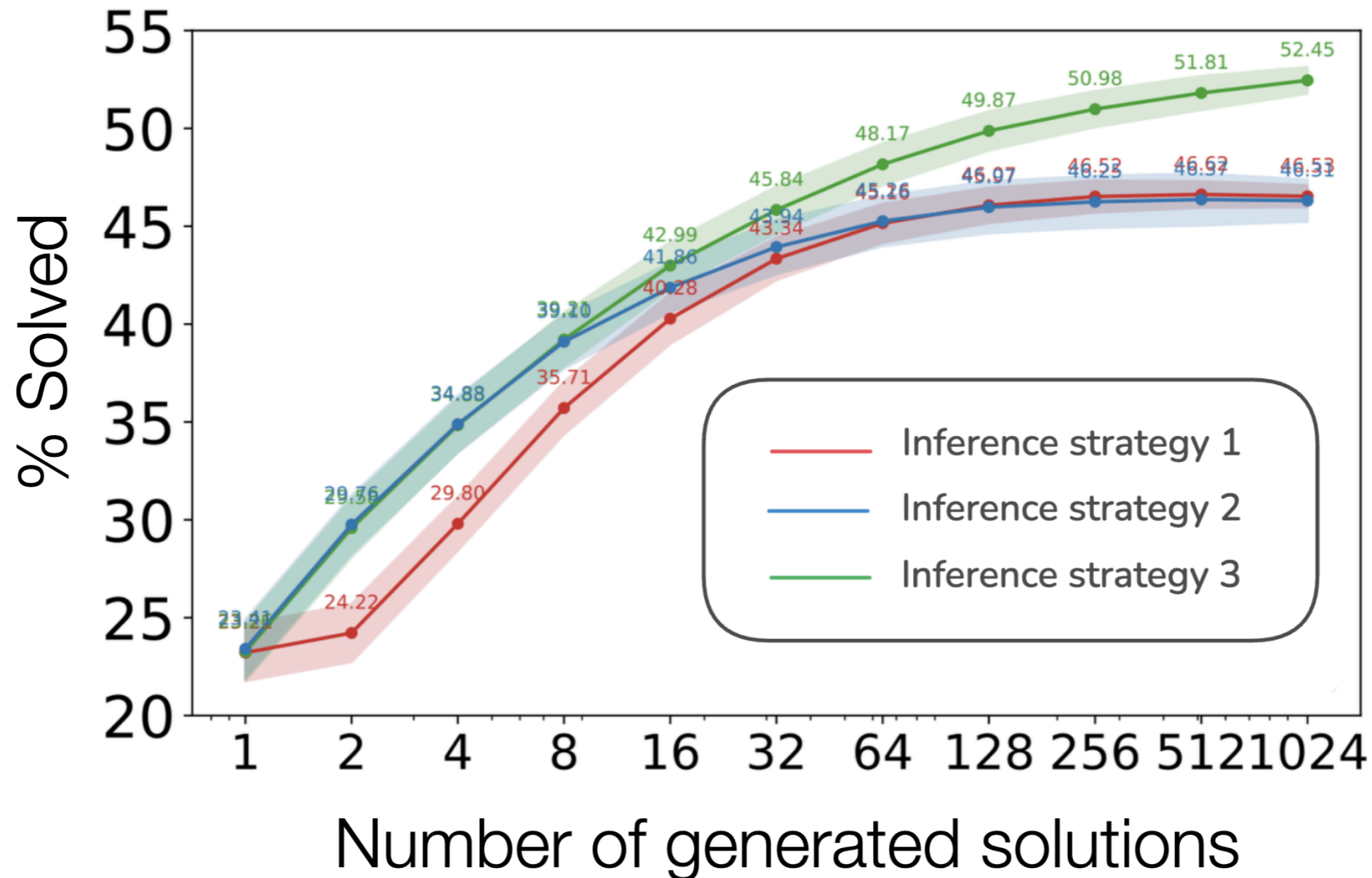


(b) Unlimited attempts per problem

[Deepmind 2022]

# Strategy 1: generate multiple times

## MATH Benchmark



# Strategy 2: generate longer outputs

input -> answer

Model Output

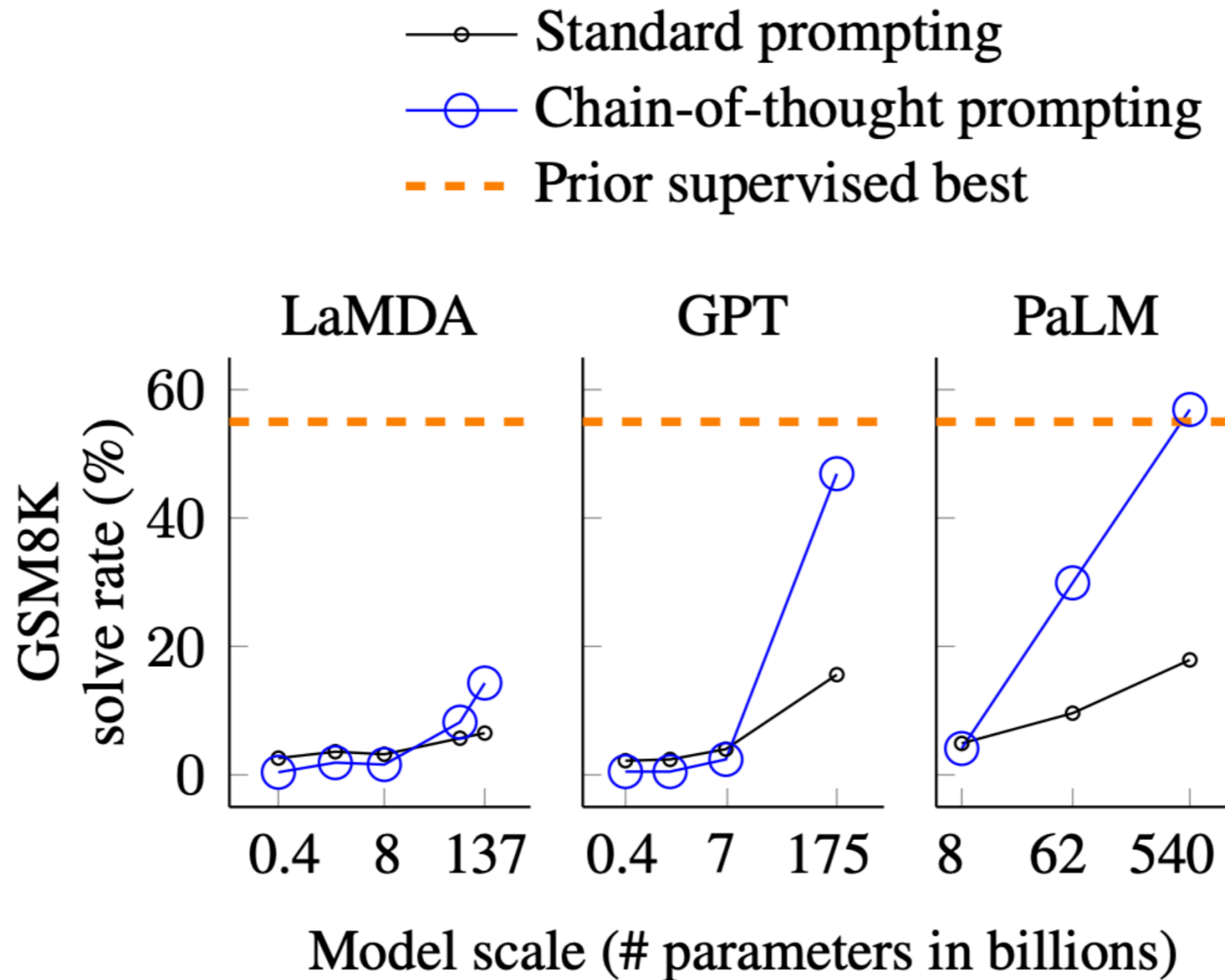
A: The answer is 27. ❌

input -> **thought**, answer

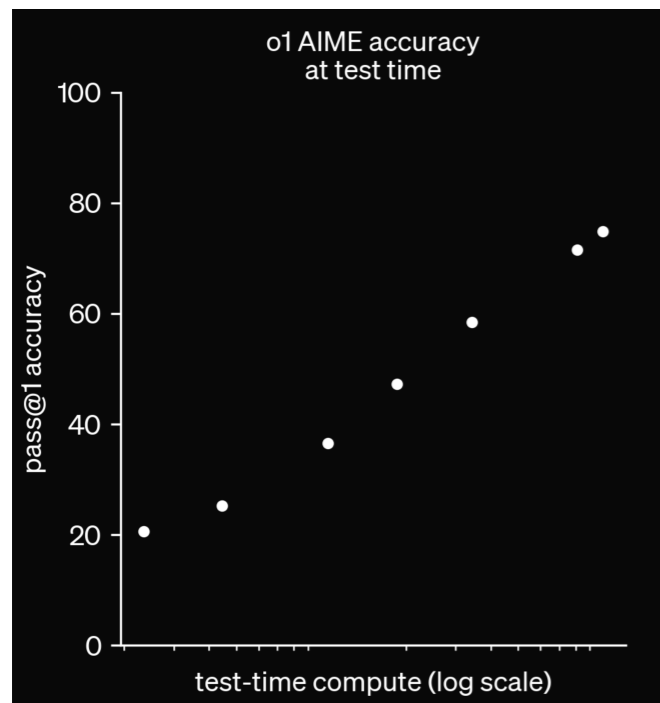
Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had  $23 - 20 = 3$ . They bought 6 more apples, so they have  $3 + 6 = 9$ . The answer is 9. ✅

# Strategy 2: generate longer outputs



# Strategy 2: generate longer outputs



o1 [OpenAI 2024]

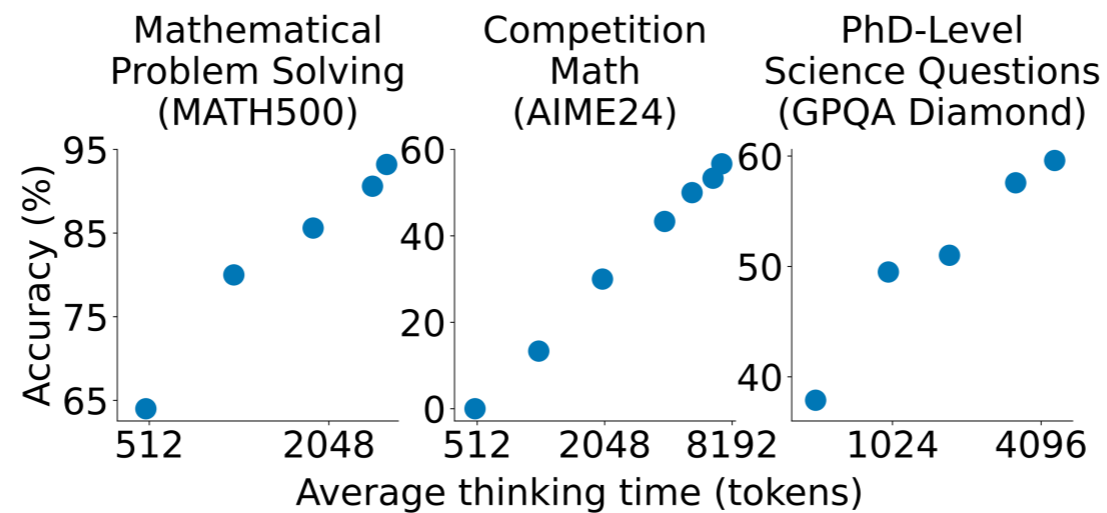
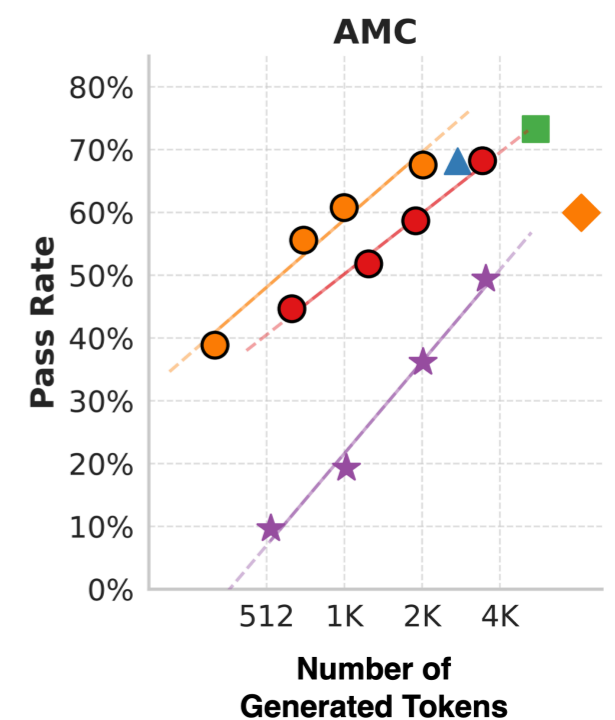


Figure 1. Test-time scaling with s1-32B. We benchmark s1-32B on reasoning-intensive tasks and vary test-time compute.

s1 [Muennighoff et al 2025]



L1 [Aggarwal & Welleck 2025]

# Today's lecture: advanced inference

- Test-time scaling strategies
  - Generating multiple times
  - Generating longer outputs

# Today's lecture: advanced inference

- Test-time scaling strategies
  - **Generating multiple times**
  - Generating longer outputs

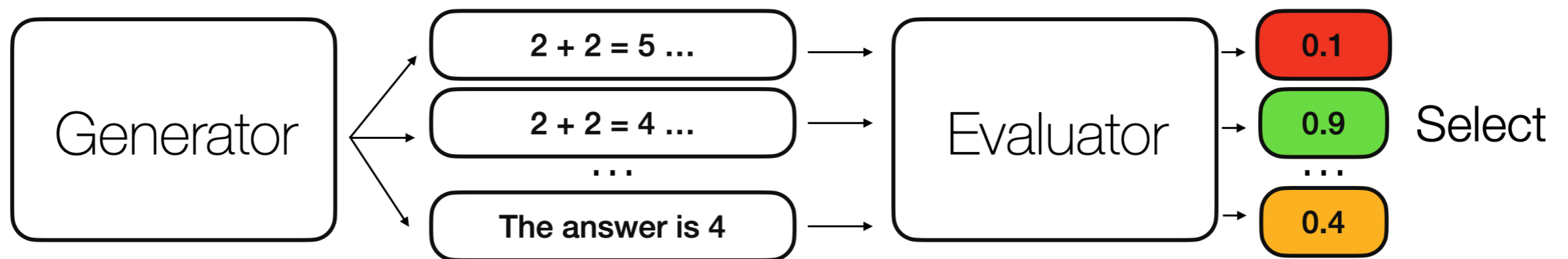
# Recap: generation/decoding algorithms

- Generator: generates a sequence with a language model
  - Example: calling an LLM API
  - Decoding algorithms
    - Greedy decoding, temperature sampling, etc.



# Meta-generation algorithms

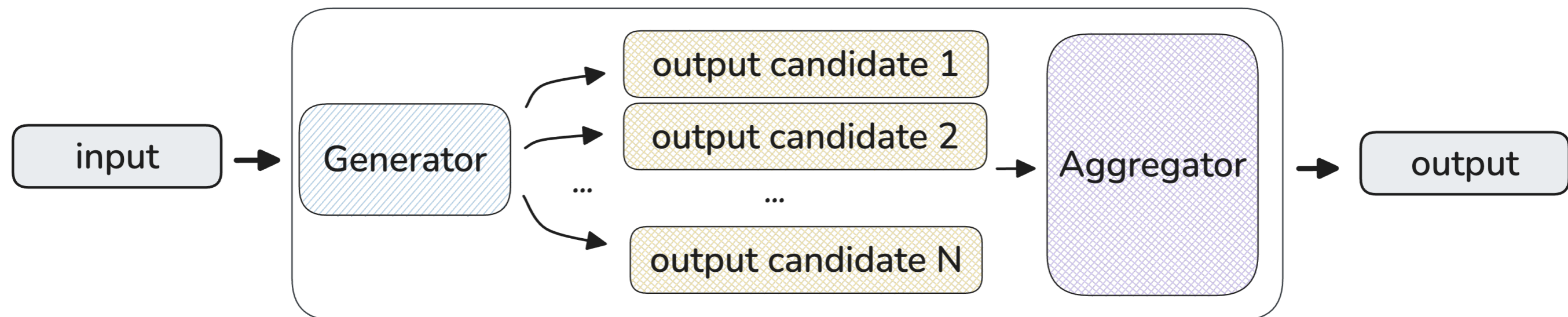
- Strategies for calling a generator multiple times



- Common patterns
  - Parallel
  - Refinement
- Others we won't discuss: tree search, hybrid strategies

# Parallel generation

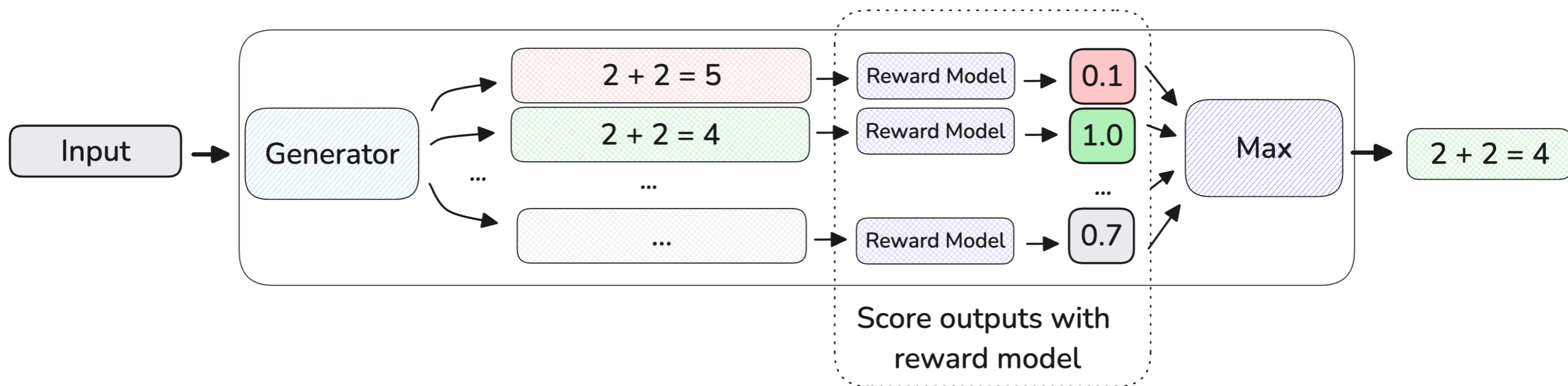
- Generate multiple candidates and aggregate them
  - Generate:
    - $\{y^{(1)}, \dots, y^{(K)}\} \sim G(\cdot | x)$
  - Aggregate:
    - $y = h(y^{(1)}, \dots, y^{(N)})$



# Parallel generation: Best-of-N

- Aggregation: **max**
- Scores: **reward model**  $v$  (aka evaluator, value model, verifier, ...)

$$\arg \max_{y^{(i)}} v(y^{(i)})$$



# Example: solving a math problem

Input:

x: Let  $f(r) = \sum_{j=2}^{2008} \frac{1}{j^r} = \frac{1}{2^r} + \frac{1}{3^r} + \dots + \frac{1}{2008^r}$ . Find  $\sum_{k=2}^{\infty} f(k)$ .

---

LLEMMA 34B solution:

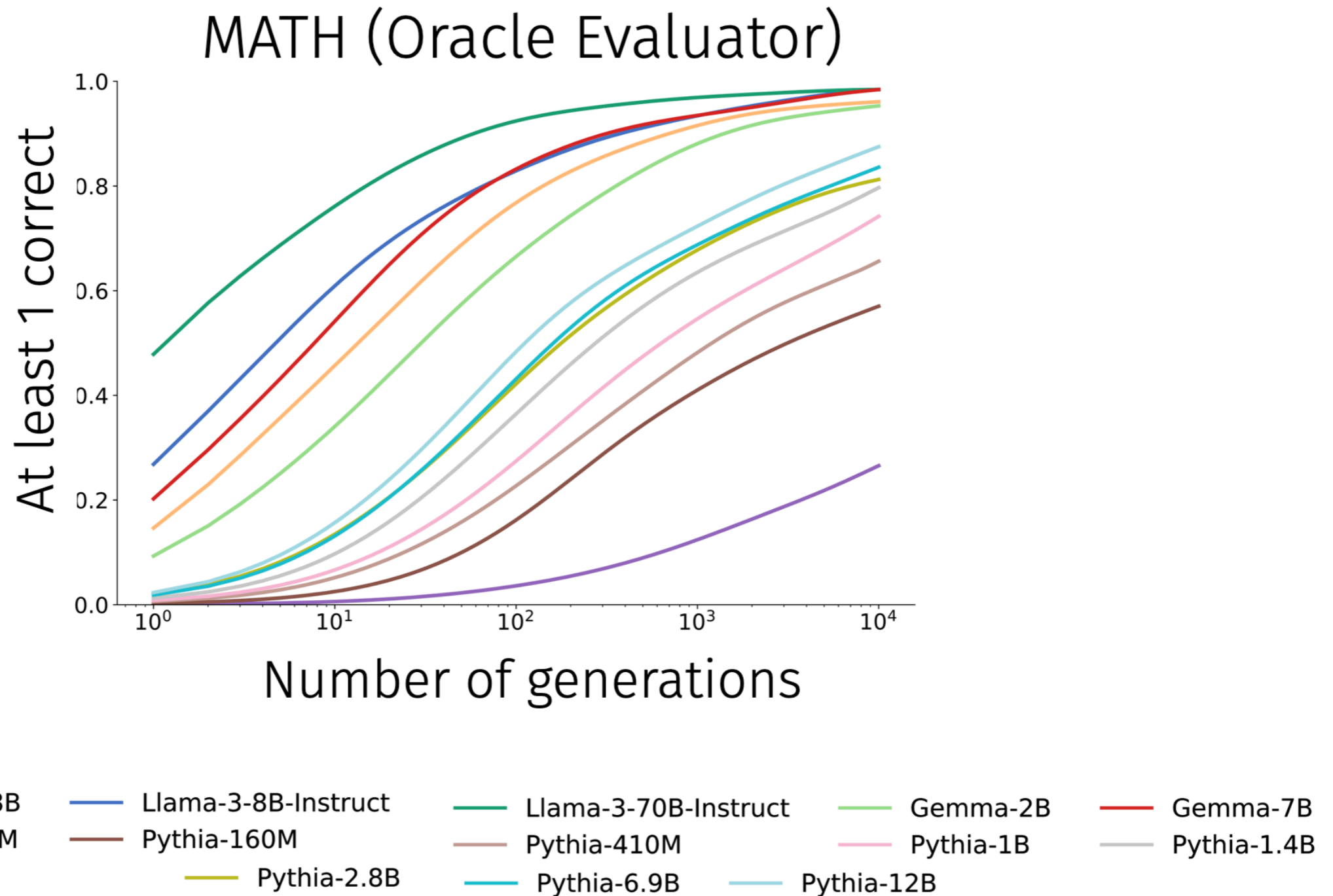
We have

z:

$$\begin{aligned} \sum_{k=2}^{\infty} f(k) &= \sum_{k=2}^{\infty} \sum_{j=2}^{2008} \frac{1}{j^k} = \sum_{j=2}^{2008} \sum_{k=2}^{\infty} \frac{1}{j^k} = \sum_{j=2}^{2008} \frac{1}{j^2} \sum_{k=0}^{\infty} \frac{1}{j^k} = \sum_{j=2}^{2008} \frac{1}{j^2} \frac{1}{1 - \frac{1}{j}} \\ &= \sum_{j=2}^{2008} \frac{1}{j(j-1)} = \sum_{j=2}^{2008} \left( \frac{1}{j-1} - \frac{1}{j} \right) \\ &= \left( \frac{1}{1} - \frac{1}{2} \right) + \left( \frac{1}{2} - \frac{1}{3} \right) + \dots + \left( \frac{1}{2007} - \frac{1}{2008} \right) \\ &= 1 - \frac{1}{2008} \\ &= \boxed{\frac{2007}{2008}}. \end{aligned}$$

y: Final Answer: The final answer is  $\frac{2007}{2008}$ .

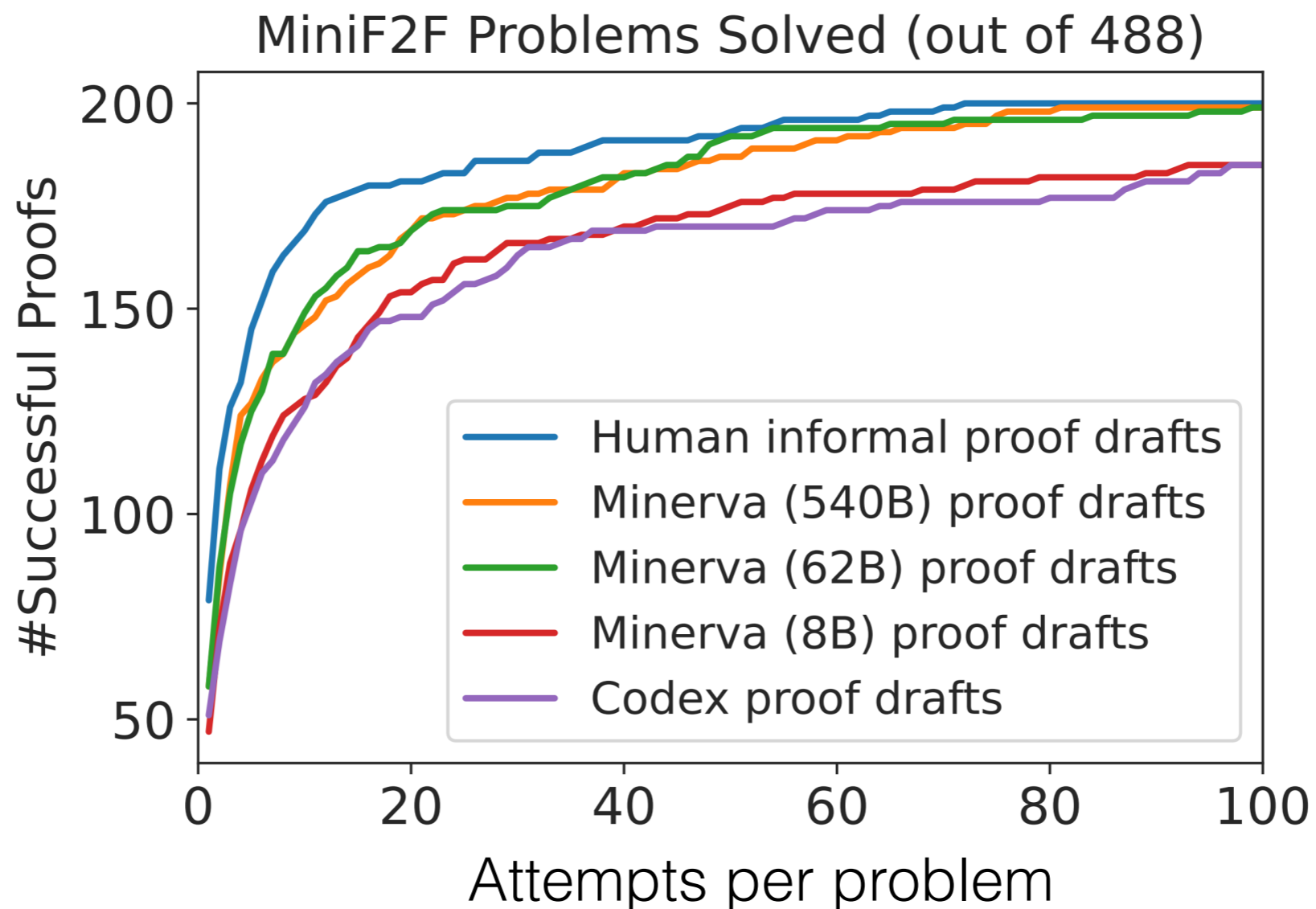
# What if we had a perfect verifier?



[Brown et al 2024]

# In some applications we have perfect verifiers

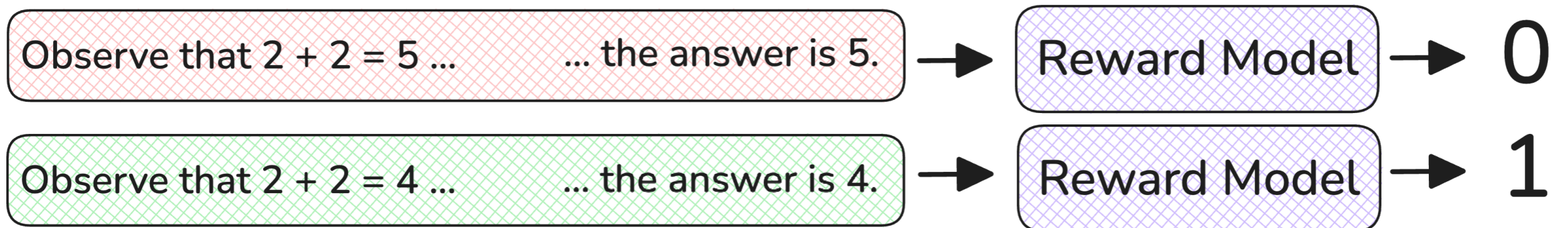
- Formal theorem proving



Draft, Sketch, and Prove [Jiang et al 2023]

# Learned reward model

- $v(y) \rightarrow \mathbb{R} \approx R(y)$
- Example: train a model to classify whether a solution is correct or incorrect



# Learned reward model

- $v(y) \rightarrow \mathbb{R} \approx R(y)$
- Example: train a model to assign a higher score to a preferred output
  - See RL for LLMs lecture!

Hello, you are awesome

>

Hello, you are #&@#\*#@#

# Example: Cobbe et al 2021 (OpenAI)

- Trains a verifier (reward model) to classify whether a solution to a grade-school math word problem (GSM8K) is correct

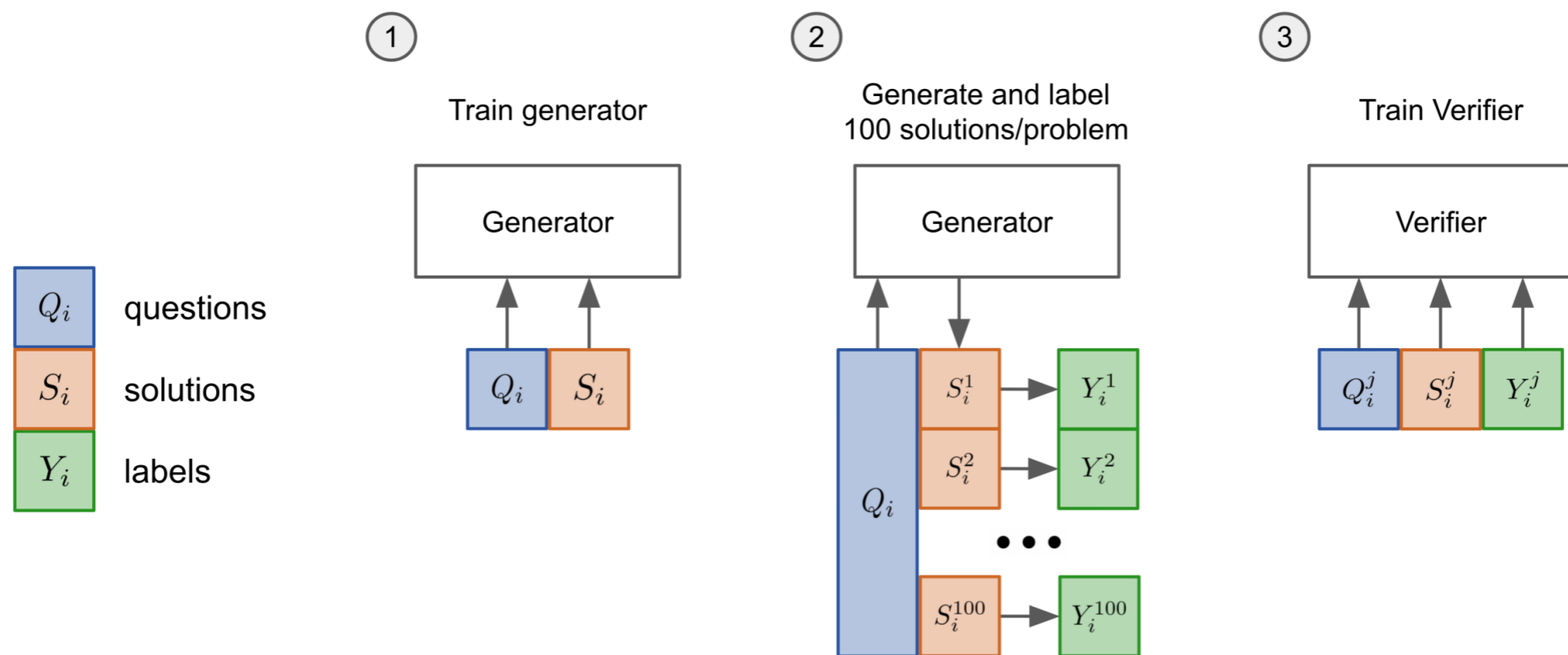
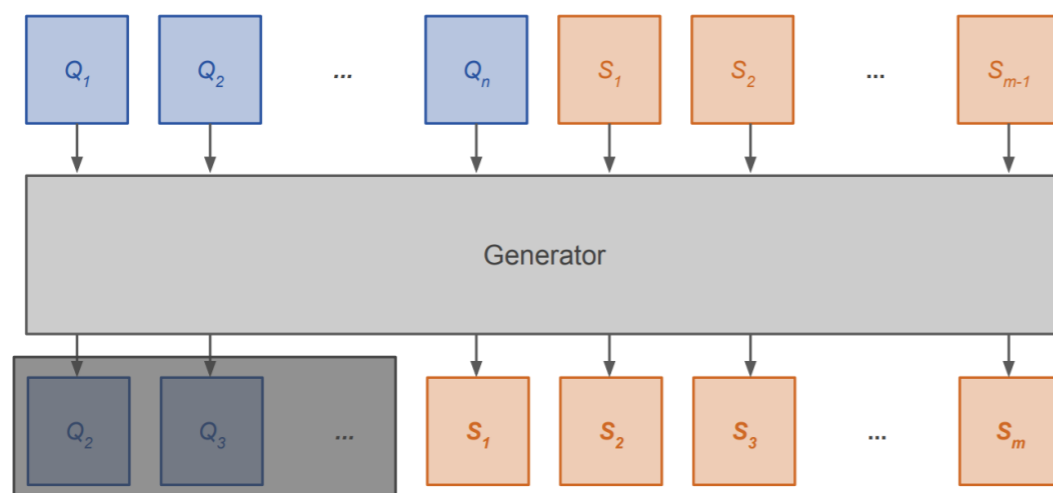


Figure 4: A diagram of the verification training pipeline.

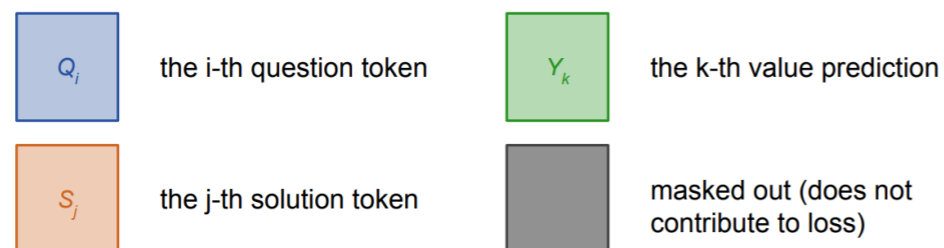
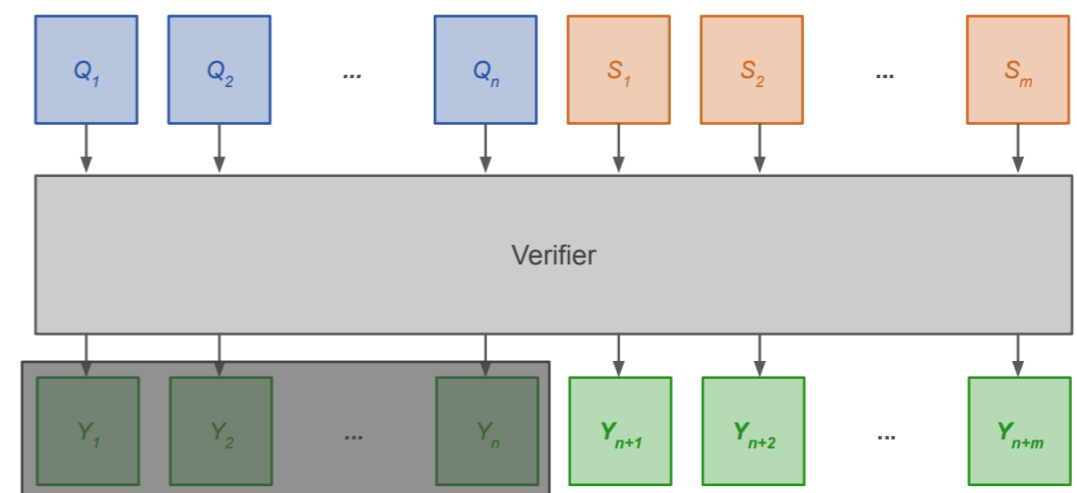
# Example: Cobbe et al 2021 (OpenAI)

- Trains a verifier (reward model) to classify whether a solution to a grade-school math word problem (GSM8K) is correct

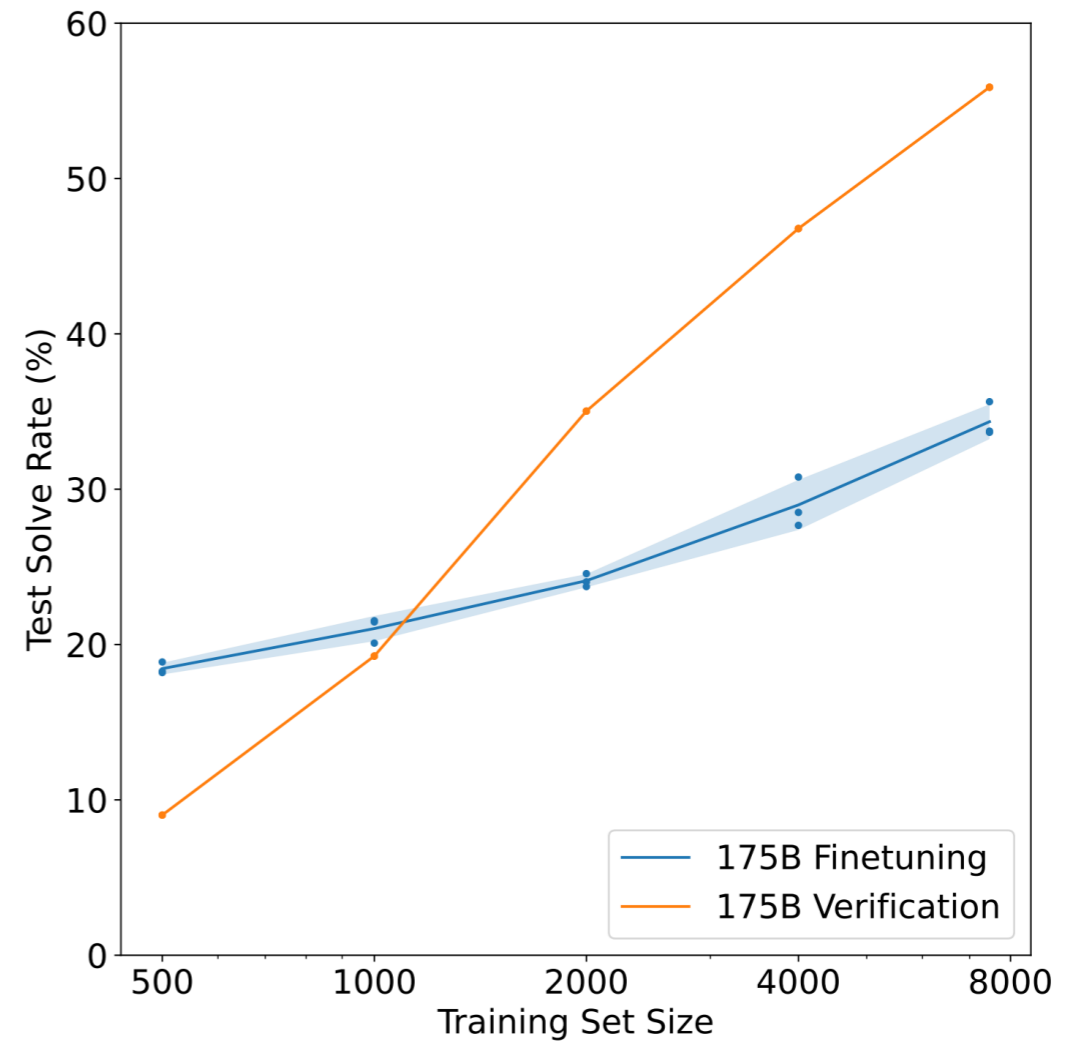
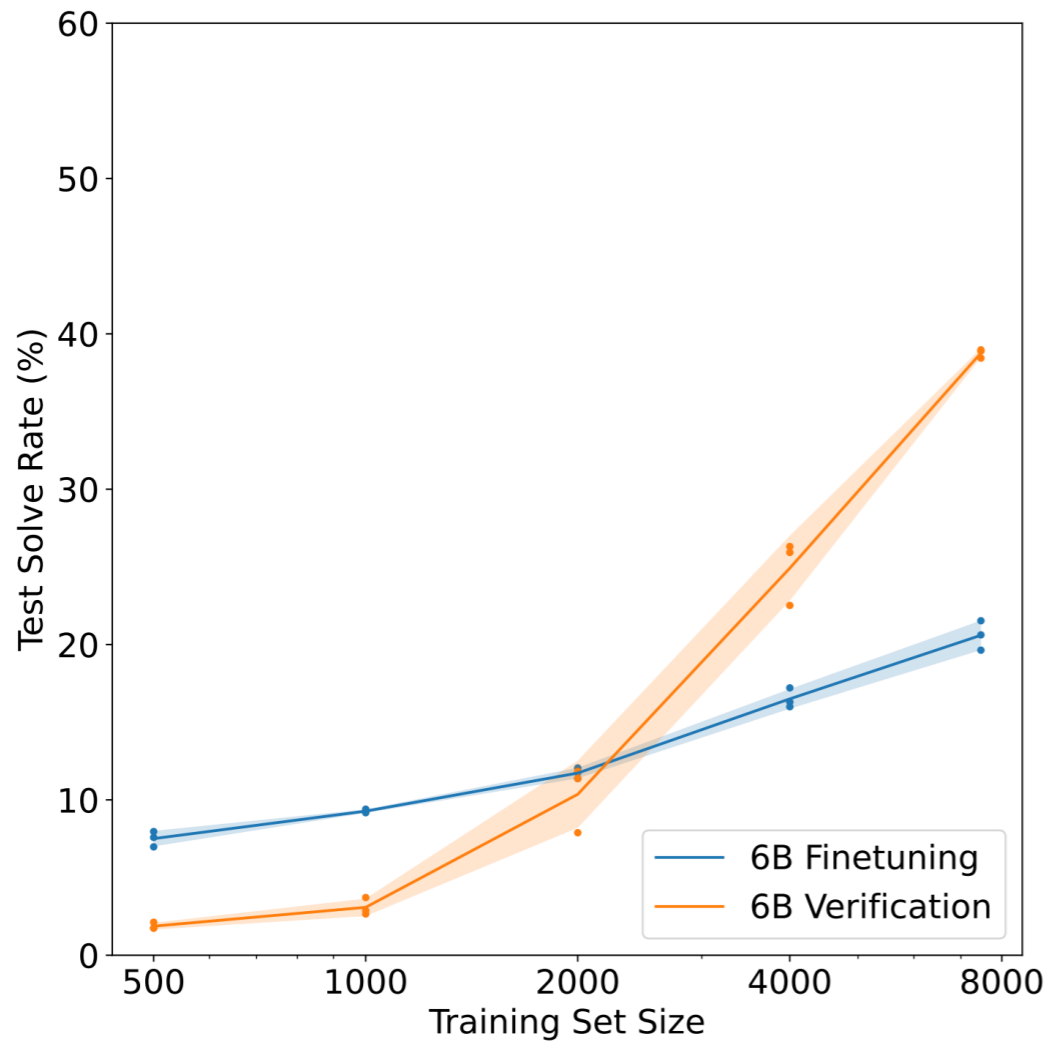
Language Modeling Objective



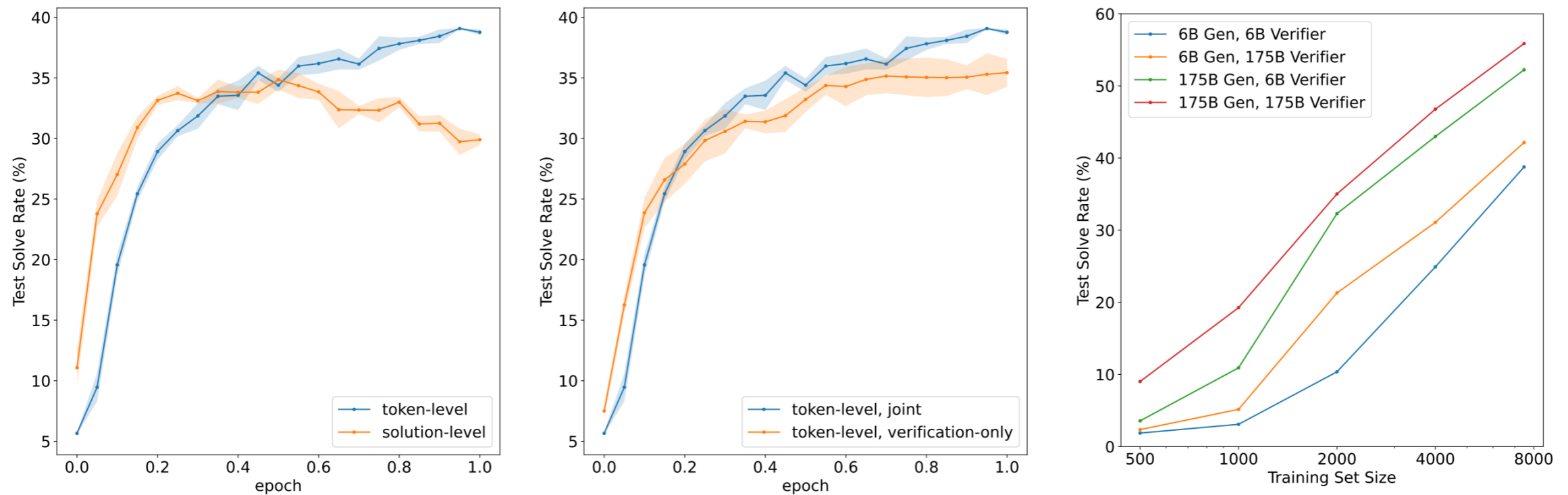
Verifier Objective



# Example: Cobbe et al 2021 (OpenAI)



# Example: Cobbe et al 2021 (OpenAI)



(a) Comparison between a verifier trained to predict correctness after every token (token-level) and one trained to predict correctness after only the final token (solution-level)

(b) Comparison between a verifier trained jointly to predict correctness and perform language modeling (joint) and one trained only to predict correctness (verification-only)

(c) Performance when varying the size of the generator and the verifier in isolation. Increasing the size of the generator has a larger impact than increasing the size of the verifier.

Figure 6: Verification ablations

Question & Model Generation	Verifier Pred.	Actual Score
<p>Claire makes a 3 egg omelet every morning for breakfast. How many dozens of eggs will she eat in 4 weeks?</p> <p>Claire makes 3 omelets every morning, so she eats <math>3 \times 7 = 21</math> omelets per week</p> <p>She eats 21 omelets per week so over 4 weeks she will eat <math>4 \times 21 = 84</math> omelets</p> <p>There are 12 in 1 dozen so she will eat <math>84 / 12 = 7</math> dozens of eggs</p> <p>### 7</p>	✓	✓
<p>Sophia and Rose went together to the market to buy onions and potatoes. Rose bought 4 times the number of onions and potatoes Sophia bought. If Rose bought 12 onions and 4 potatoes, how many onions and potatoes in total did Sophia buy at the market?</p> <p>If Rose bought 12 onions and 4 potatoes, she bought <math>12 + 4 = 16</math> onions and potatoes.</p> <p>If Rose bought 4 times the number of onions and potatoes that Sophia bought, Sophia bought <math>16 / 4 = 4</math> onions and potatoes at the market.</p> <p>### 4</p>	✗	✓
<p>Two sisters, Elizabeth and Margareth, bought beads. Elizabeth bought 1 pack of red and 2 packs of clear beads, while Margareth bought 3 packs of blue and 4 packs of red beads. How many more beads do one sister have than the other, if each pack of beads contains 20 pieces of beads?</p> <p>Elizabeth bought 1 pack of red and 2 packs of clear, totaling <math>1 + 2 = 3</math> packs of beads.</p> <p>Margareth bought 3 packs of blue and 4 packs of red, totaling <math>3 + 4 = 7</math> packs of beads.</p> <p>So, Elizabeth has <math>3 - 7 = -4</math> more packs of beads than Margareth.</p> <p>And since each pack of beads contains 20 pieces of beads, Elizabeth has <math>20 \times 4 = 80</math> more beads than Margareth.</p> <p>### 80</p>	✗	✓
<p>Howard spends \$8 dollars at the arcade on Monday. On Tuesday, he spends twice as much at the arcade as he did on Monday. On Wednesday, he spends 4 times as much at the arcade as he spent on Tuesday. If he originally had \$100, how much money does he have left?</p> <p>He spent \$8 dollars on Monday and <math>8 \times 2 = 16</math> dollars on Tuesday.</p> <p>He spent \$16 dollars on Tuesday and <math>16 \times 4 = 64</math> dollars on Wednesday.</p> <p>He has \$100 dollars and spent \$64 dollars so he has <math>100 - 64 = 36</math> dollars left.</p> <p>### 36</p>	✗	✗
<p>In a jewelers store, the price of a gold Jewell is <math>\frac{4}{5}</math> times as much as the price of a diamond Jewell. The cost of a silver Jewell is \$400 less than the price of gold. If a diamond Jewell is \$2000, find the total price for all three jewels.</p> <p>The price of a gold Jewell is <math>\frac{4}{5} \times \\$2000 = 1600</math>.</p> <p>The cost of a silver Jewell is <math>\\$2000 - \\$400 = 1600</math>.</p> <p>The total price for all the jewels is <math>\\$2000 + \\$1600 + \\$1600 = 5200</math></p> <p>### 5200</p>	✓	✗

# Why best-of-N?

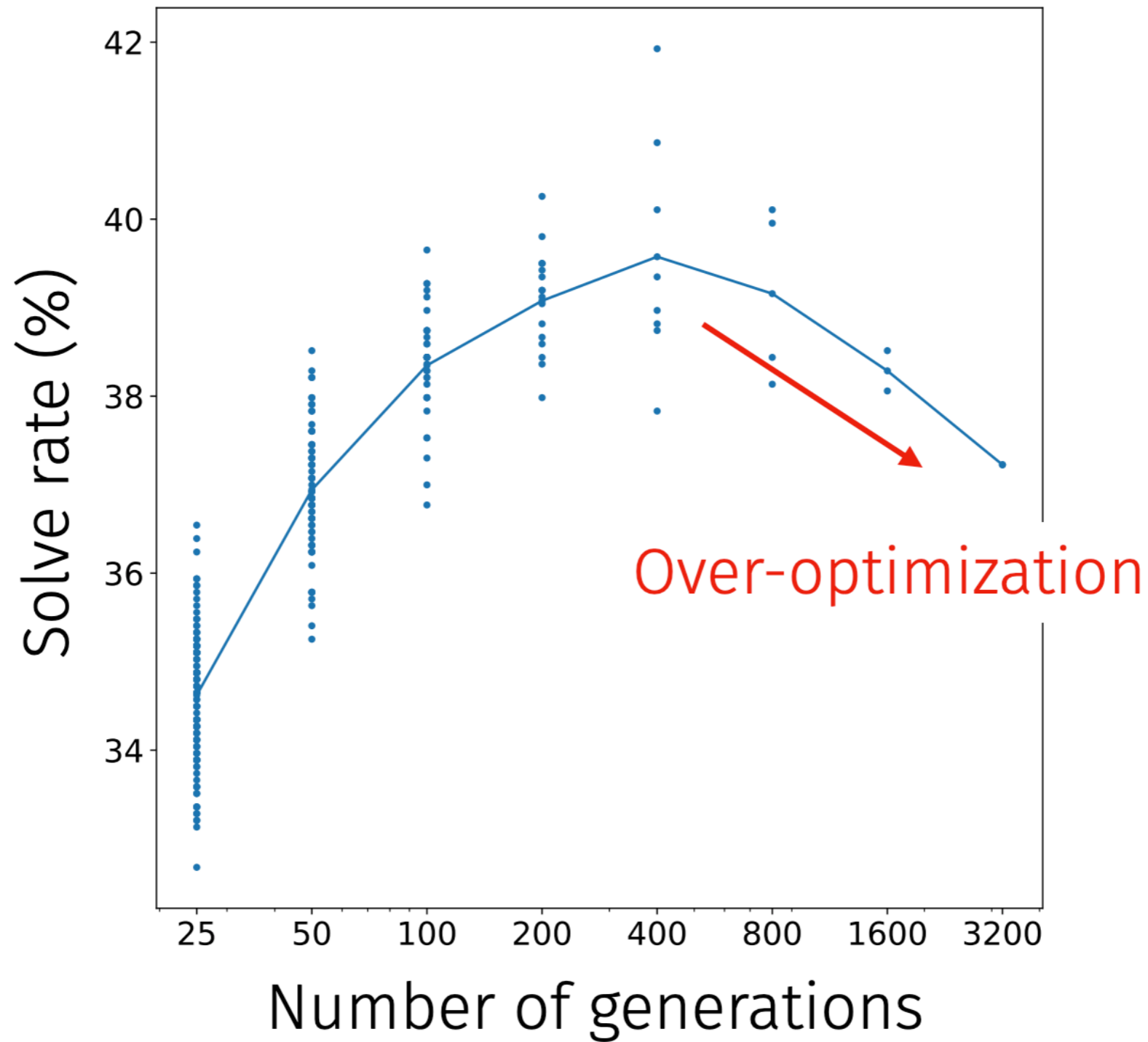
- Approximates the maximum true reward:

$$\text{Best-of-}N = \arg \max_{y \in \{y^{(1)}, \dots, y^{(N)}\}} v(y)$$

$$\approx \arg \max_y v(y) \longleftarrow \text{Gets better as } N \text{ increases!}$$

$$\approx \arg \max_y R(y) \longleftarrow \text{Suffers from imperfect reward model, aka "Over-optimization" / reward hacking}$$

# Over-optimization / reward hacking






[Cobbe et al 2021]



# Improving the reward model: process labels




- If we can get good labels for intermediate steps, it's possible to train a better reward model
  - Getting good labels is difficult in general!




# Example: [Lightman et al 2023] (OpenAI)

The denominator of a fraction is 7 less than 3 times the numerator. If the fraction is equivalent to  $\frac{2}{5}$ , what is the numerator of the fraction? (Answer: )

   Let's call the numerator  $x$ .

   So the denominator is  $3x-7$ .

   We know that  $\frac{x}{3x-7} = \frac{2}{5}$ .

   So  $5x = 2(3x-7)$ .

    $5x = 6x - 14$ .

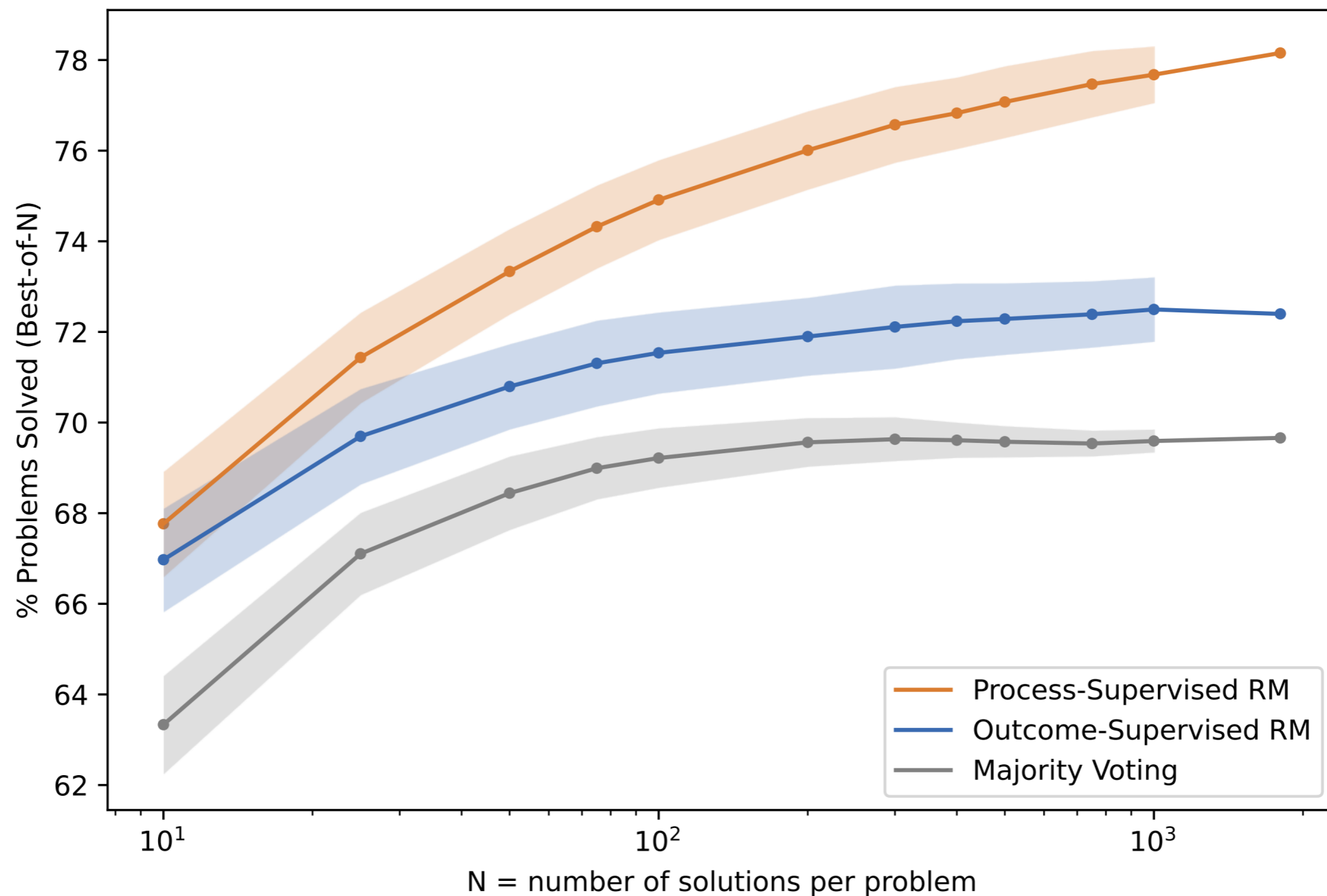
   So  $x = 7$ .

Figure 1: A screenshot of the interface used to collect feedback for each step in a solution.

Human-annotated process labels

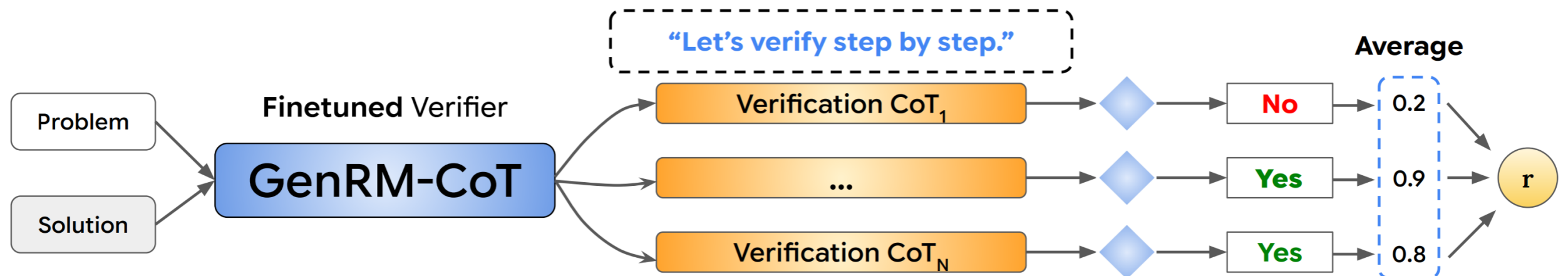
# Example: [Lightman et al 2023] (OpenAI)

	ORM	PRM	Majority Voting
% Solved (Best-of-1860)	72.4	<b>78.2</b>	69.6



# Improving the reward model: CoT

- Studies have found that using chain-of-thought tends to help for the reward model



# Improving the reward model: CoT

- Studies have found that using chain-of-thought tends to help for the reward model

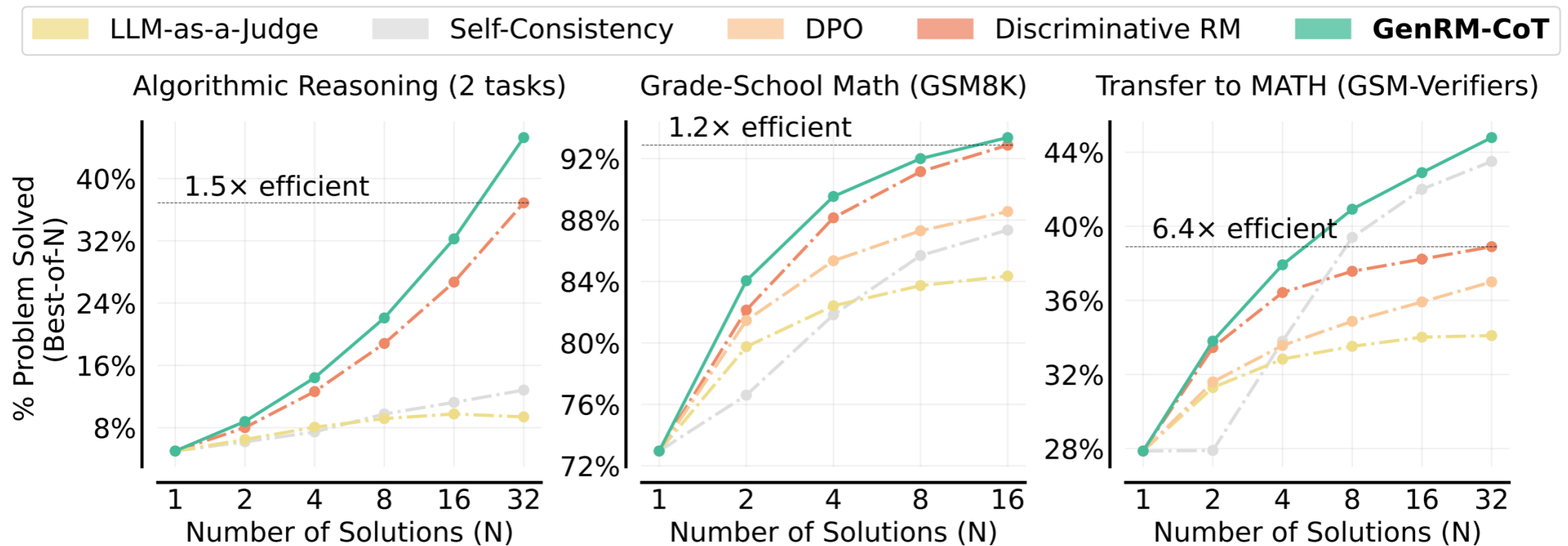
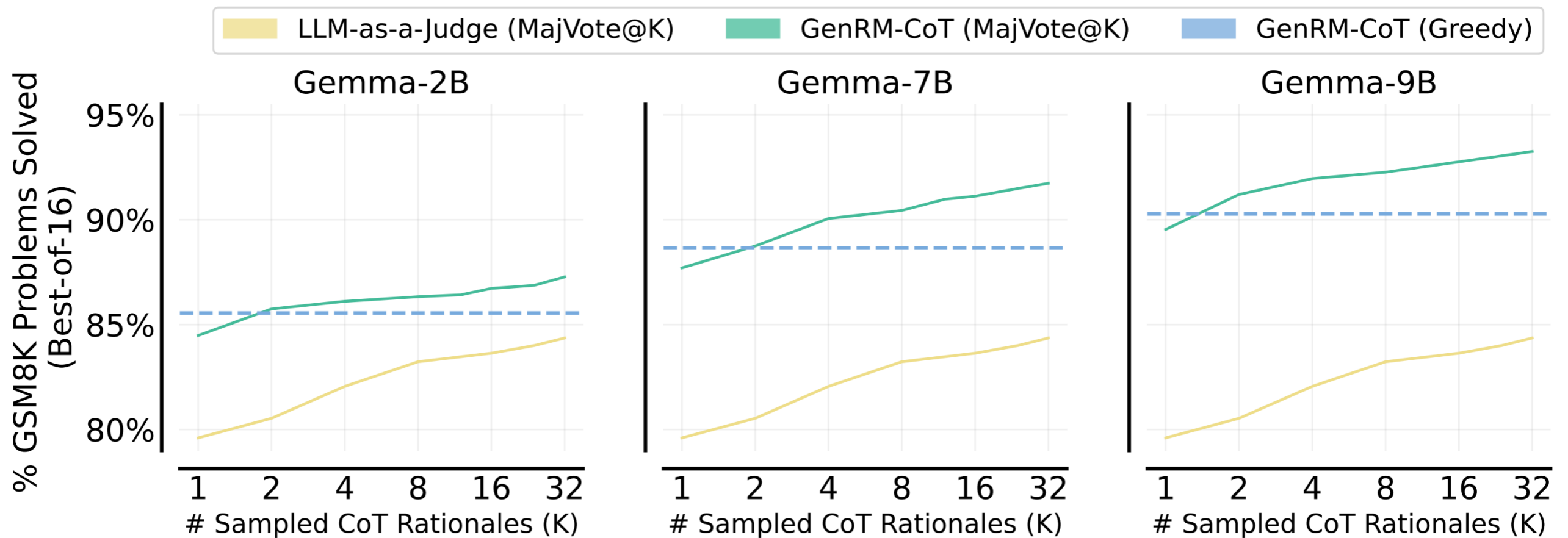


Figure 5 | **Sample-Efficient Scaling with Generative Verifiers.** GenRM-CoT outperforms other methods, especially for length generalization on algorithmic tasks (Gemma-2B verifiers) and easy-to-hard generalization on MATH (Gemma2-9B verifiers). Specifically, GenRM-CoT nearly matches the oracle verifier’s Best-of-N performance on algorithmic tasks. On MATH, it matches discriminative verifier’s Best-of-32 performance using **6.4× fewer solutions**.

# Improving the reward model: CoT

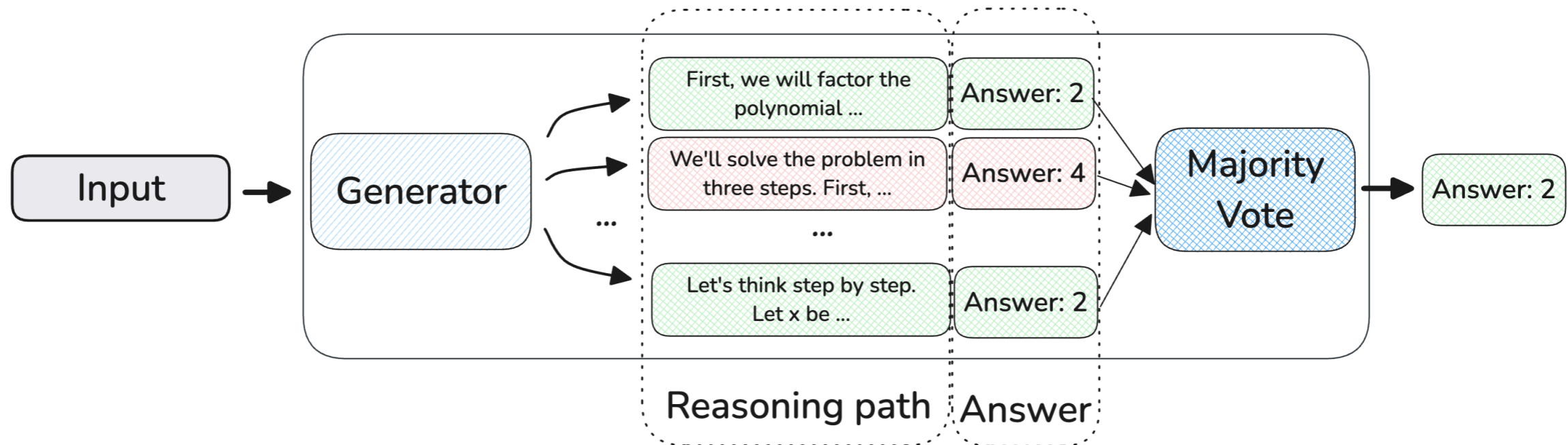
- Inference compute used for evaluation can itself be scaled up



# Voting (aka self-consistency)

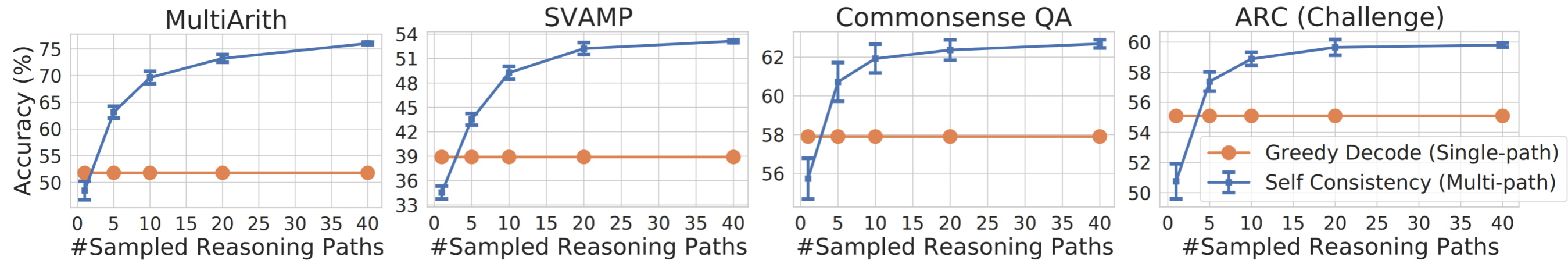
- Aggregation: pick the most common answer

$$\arg \max_a \sum_{i=1}^N 1[y^{(i)}=a]$$



- Pro: No reward model needed!
- Potential con: need a notion of equivalence of outputs
  - (e.g., same answer)

# Voting



Self-Consistency [Wang et al 2022]

# Voting

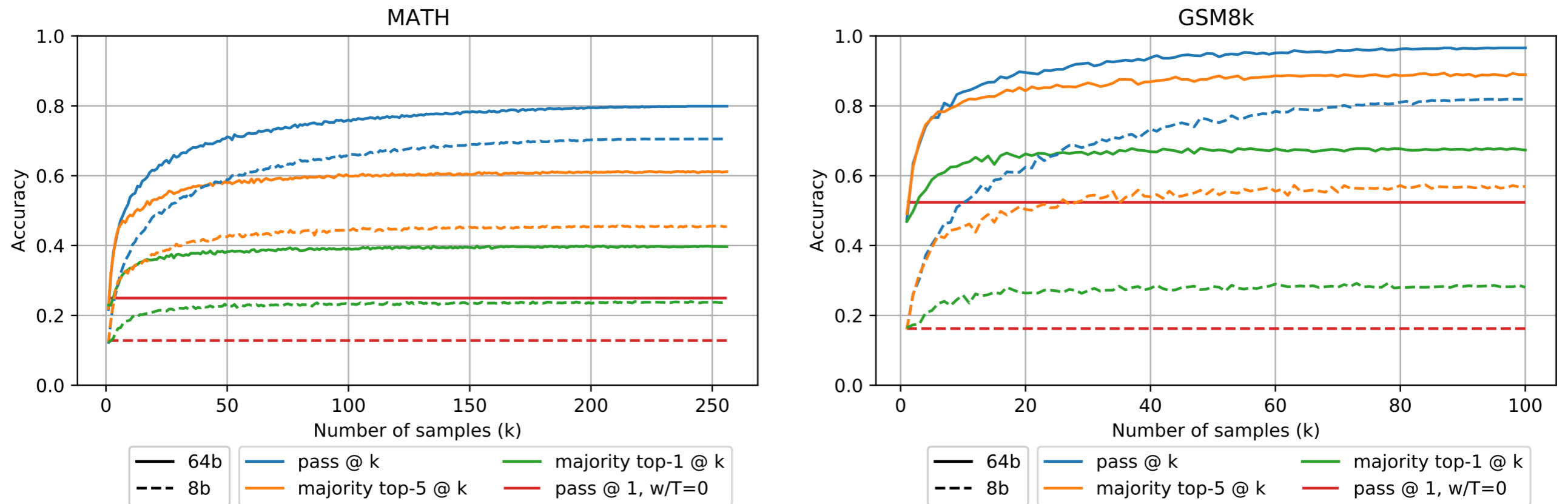


Figure 6: Accuracy as a function of  $k$ , the number of samples per task. Majority voting performance saturates quickly while `pass@k` seems to continue improving slowly. Accuracies were computed using exact string match (without SymPy processing).

# Voting

- Voting was better than best-of-N w/ model log-likelihood as the scoring function

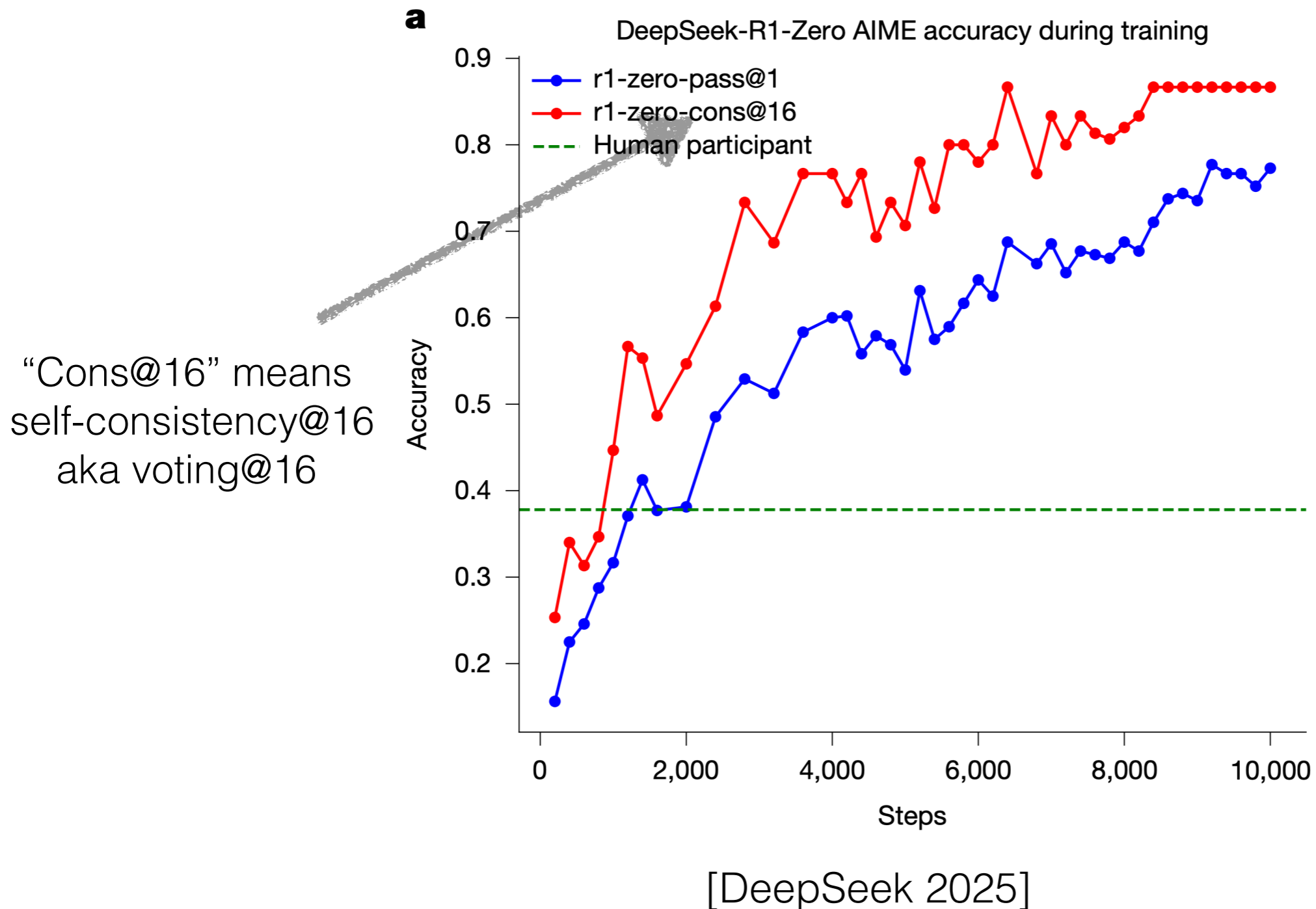
---

	MATH
Minerva 62B, <code>pass1</code> $T = 0.0$	26.5%
Minerva 62B, Majority Voting 1@k	42.0%
Minerva 62B, <code>pass1</code> $T = 0.6$	21.8%
Minerva 62B, Log-likelihood 1@k	23.8%

---

# Voting

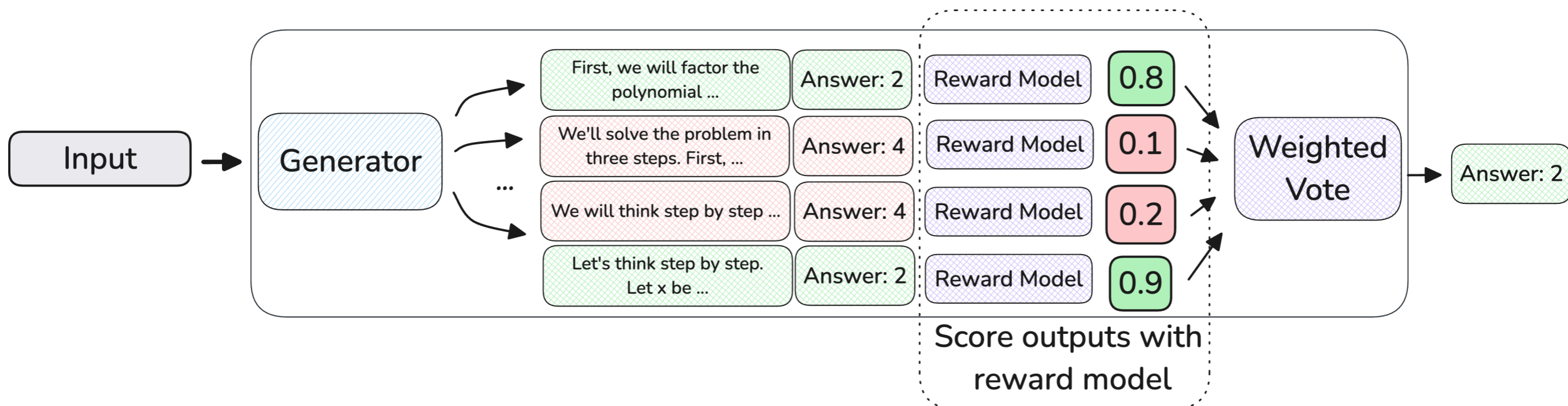
- Voting can still help with recent long CoT models like R1



# Weighted Voting

- Aggregation: pick the most common answer
- Scoring: use a reward model to weight votes

$$\arg \max_a \sum_{i=1}^N v(y^{(i)}) \mathbb{1}[y^{(i)}=a]$$



# Weighted Voting

- Can perform better than unweighted voting
- Improves with a better reward model

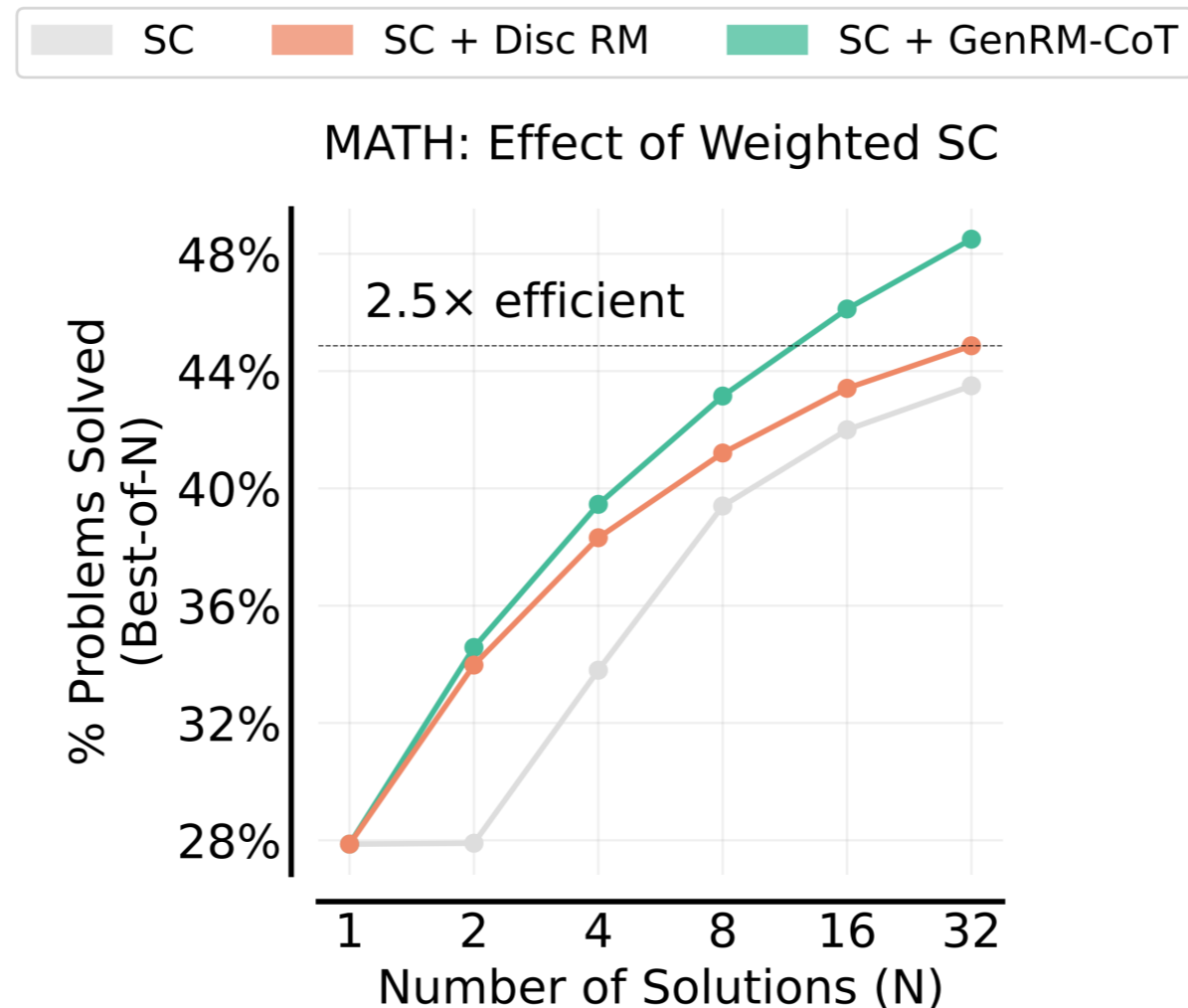


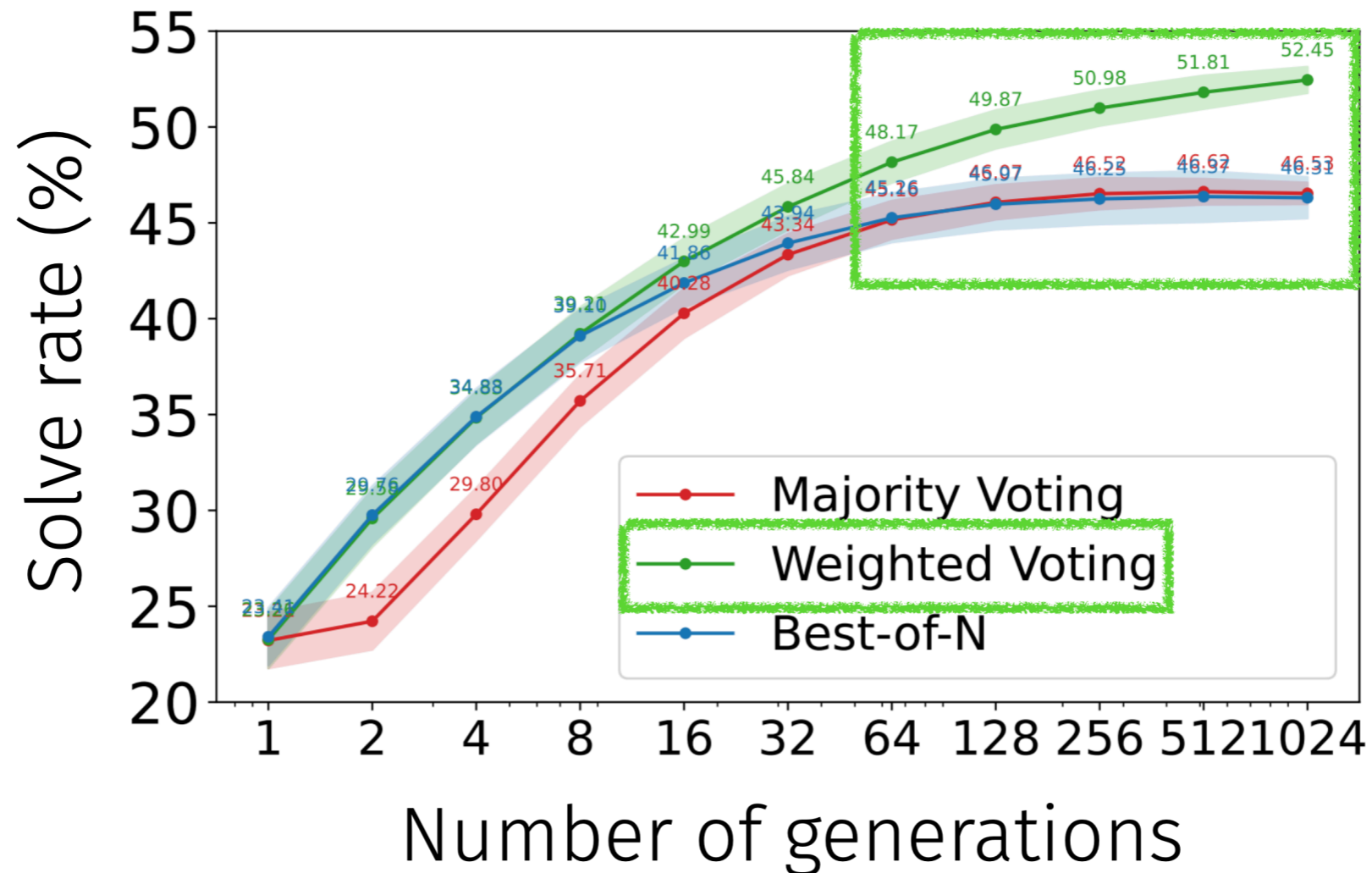
Figure 8 | **Weighted Self-Consistency** on MATH.

[Zhang et al 2025]

# Weighted Voting

- Can perform better than best-of-N

## MATH (Learned Reward)



# What is voting doing?

- When we have a chain-of-thought followed by a final output, voting “marginalizes out” the intermediate reasoning chains
- As the number of candidates  $N \rightarrow \infty$ , voting accuracy converges to:

$$\frac{1}{M} \sum_{i=1}^M \left[ a_i^* = \arg \max_a \sum_z v(x, z, a) g(z, a | x) \right]$$

Notation:

- $(x, z, a)$ : (input, solution, answer)
- $M$ : number of test examples

# What is voting doing?

- When we have a chain-of-thought followed by a final output, voting “marginalizes out” the intermediate reasoning chains
- As the number of candidates  $N \rightarrow \infty$ , voting accuracy converges to:

$$\frac{1}{M} \sum_{i=1}^M \mathbb{I} \left[ a_i^* = \arg \max_a \sum_z v(x, z, a) g(z, a | x) \right]$$

- **Takeaway 1:** when is weighted voting better than voting?
  - When  $v \cdot g$  assigns more total mass to correct answers than  $g$

# What is voting doing?

- When we have a chain-of-thought followed by a final output, voting “marginalizes out” the intermediate reasoning chains
- As the number of candidates  $N \rightarrow \infty$ , voting accuracy converges to:

$$\frac{1}{M} \sum_{i=1}^M \mathbb{I} \left[ a_i^* = \arg \max_a \sum_z v(x, z, a) g(z, a | x) \right]$$

- **Takeaway 2:** will accuracy keep improving with more samples?
  - It converges to the accuracy shown above

# What is voting doing?

- When we have a chain-of-thought followed by a final output, voting “marginalizes out” the intermediate reasoning chains
- As the number of candidates  $N \rightarrow \infty$ , voting accuracy converges to:

$$\frac{1}{M} \sum_{i=1}^M \mathbb{E} \left[ a_i^* = \arg \max_a \sum_z v(x, z, a) g(z, a | x) \right]$$

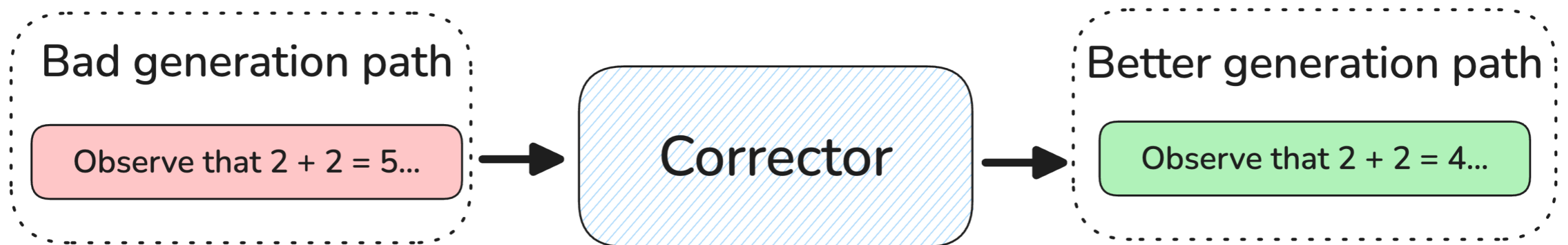
- **Takeaway 3:** how do we improve performance further?
  - Improve the reward model  $v$
  - Improve the generator  $g$  (better model or better algorithm)

# Meta-generation algorithms

- Strategies for calling a generator multiple times
- Common patterns
  - Parallel
    - Best-of-N, voting
  - ***Next: Refinement***

# Refinement

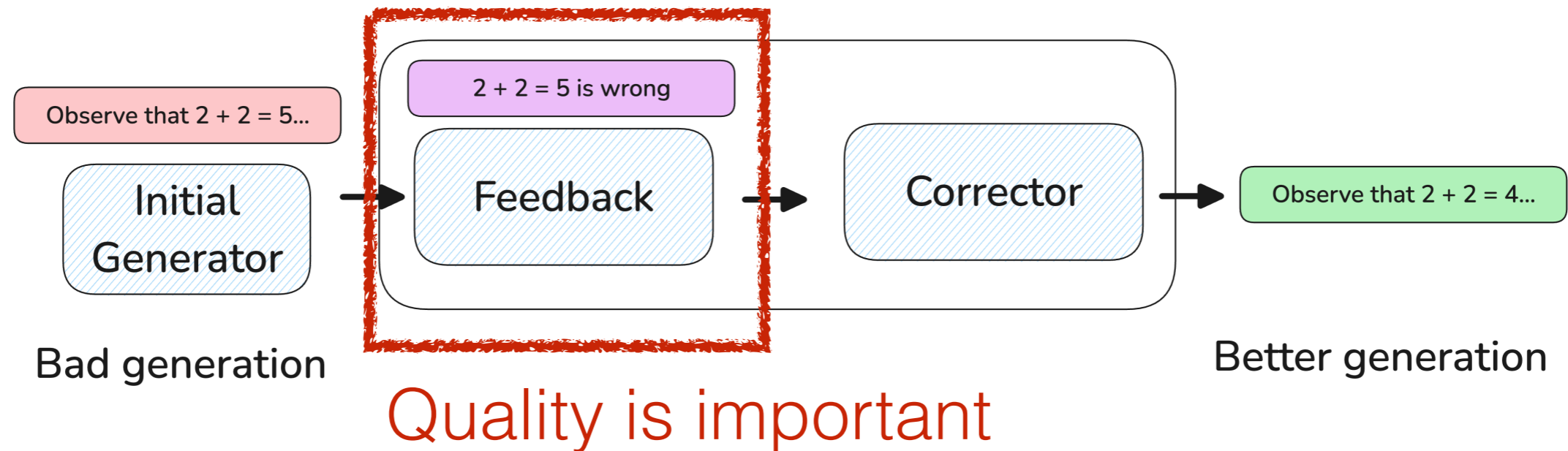
- Generate an output, feed it back into the model



- Repeat:
  - $y^{(i+1)} \sim g(x, y^{(i)})$

# Refinement

- Generate an output, receive feedback, generate a new output using the feedback



- Repeat:
  - $y^{(i+1)} \sim g(x, y^{(i)}, F(y^{(i)}))$

# Refinement

- Key question: **quality** and **source** of the feedback
- **Extrinsic**: new external information enters the refinement loop
- **Intrinsic**: no external information enters the refinement loop

# Extrinsic feedback

- Refinement tends to work well with “good” extrinsic feedback
  - Adds new information
  - High quality (e.g., accurate, noise-free)
  - Specific
    - Localizes errors
    - Gives something specific for the model to refine

# Example: code generation

- Feedback
  - Unit test results (extrinsic)
  - Explanation (intrinsic)
- $\leq 10$  refinement turns

---

SELF-DEBUGGING (this work)

---

Codex	72.2 ( $n = 10$ )
Simple	73.6
UT	75.2
UT + Expl.	<b>75.6</b>

---

## Unit Test + Explanation (+Expl.)

Below are C++ programs with incorrect Python translations. *Explain the original code, then explain the translations line by line* and correct them using the provided feedback.

[C++]

[C++ Explanation]

[Original Python]

[Python Explanation]

[UT Feedback]

[Revised Python #1]

[Python Explanation]

[UT Feedback]

[Revised Python #2]

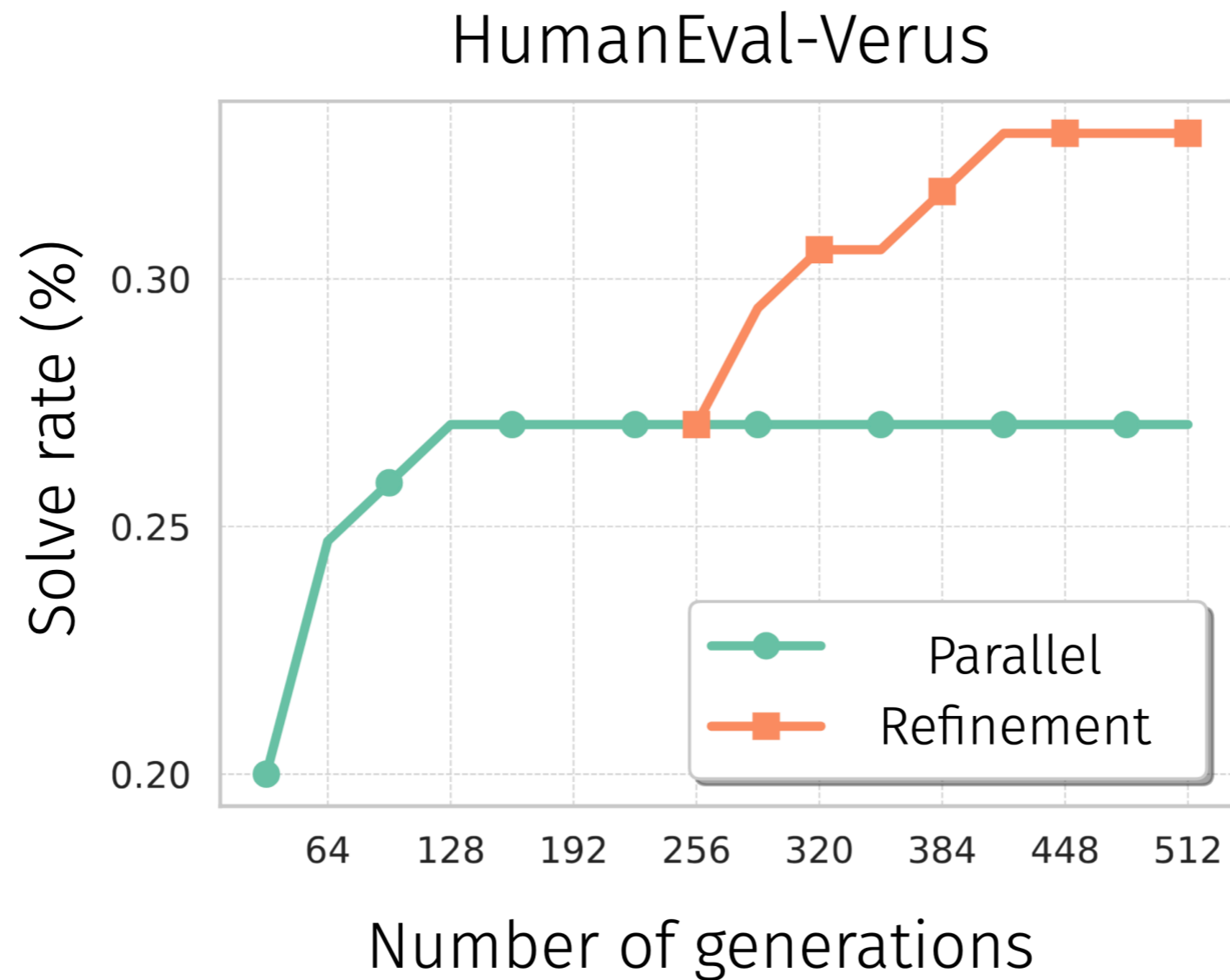
[Python Explanation]

...



# Example: verified code generation

- Feedback: formal program verifier (extrinsic)



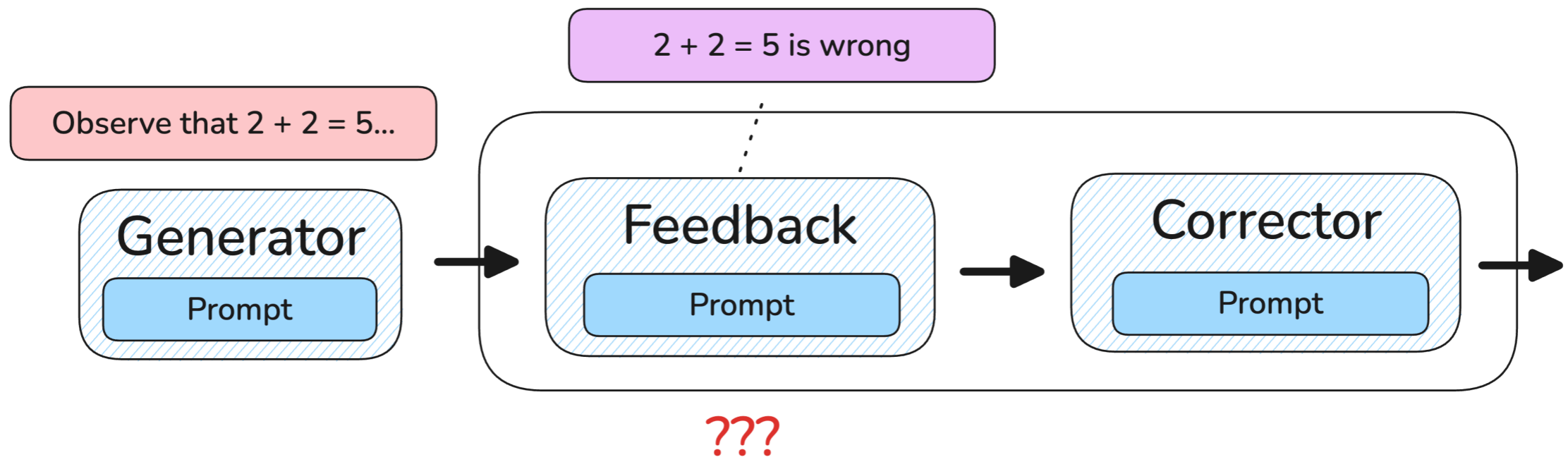
*AlphaVerus*, [Aggarwal et al 2024]

# Refinement with extrinsic feedback

- Success cases
  - Code execution / test cases [Chen et al 2024]
  - Verifiers [Aggarwal et al 2024]
  - Retrievers [Asai et al 2024]
  - Tools + agent environment
  - ...

# Intrinsic feedback

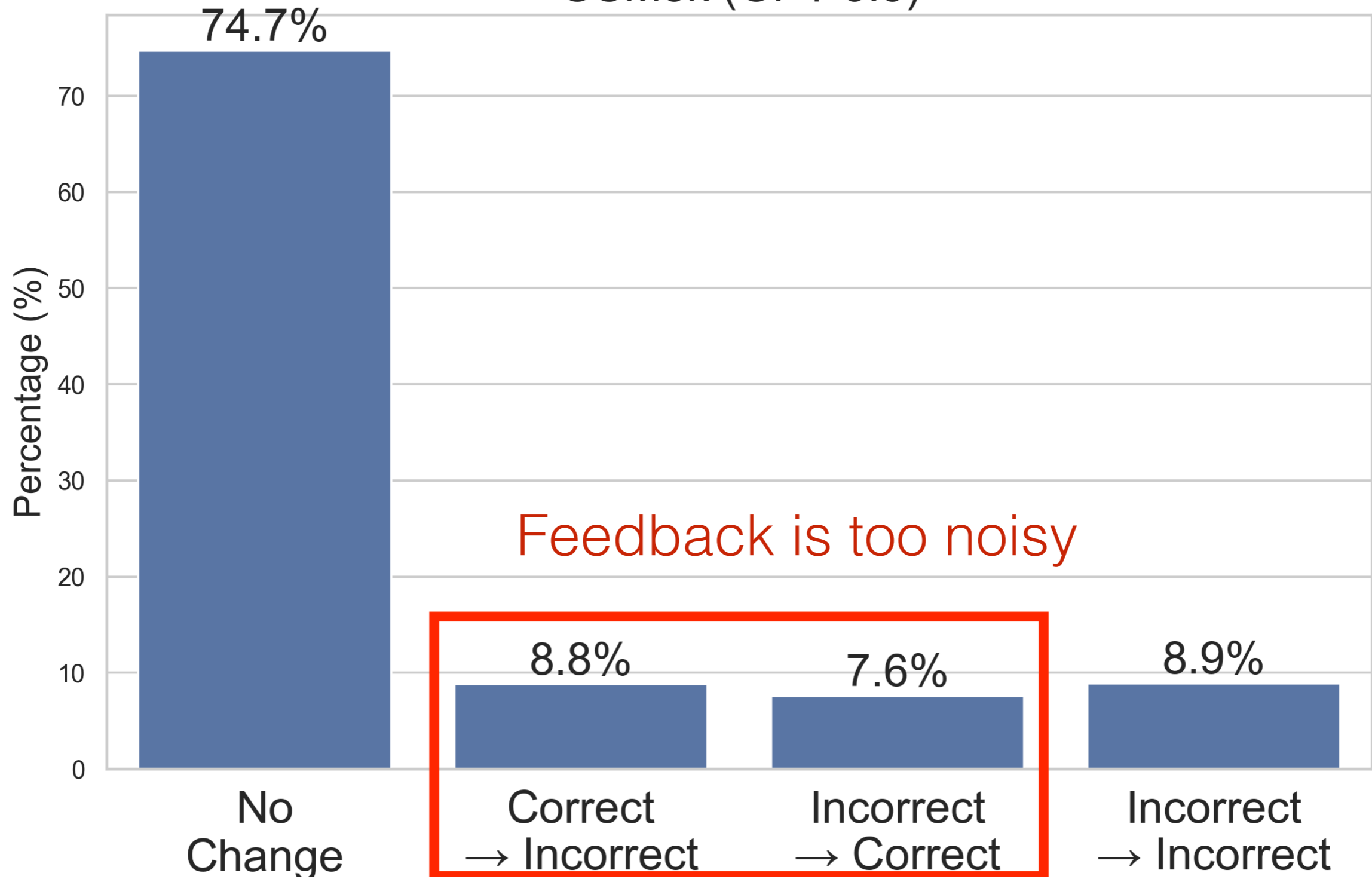
- *Self-refinement*: re-prompt the same model to generate, give feedback, and refine



# Intrinsic feedback

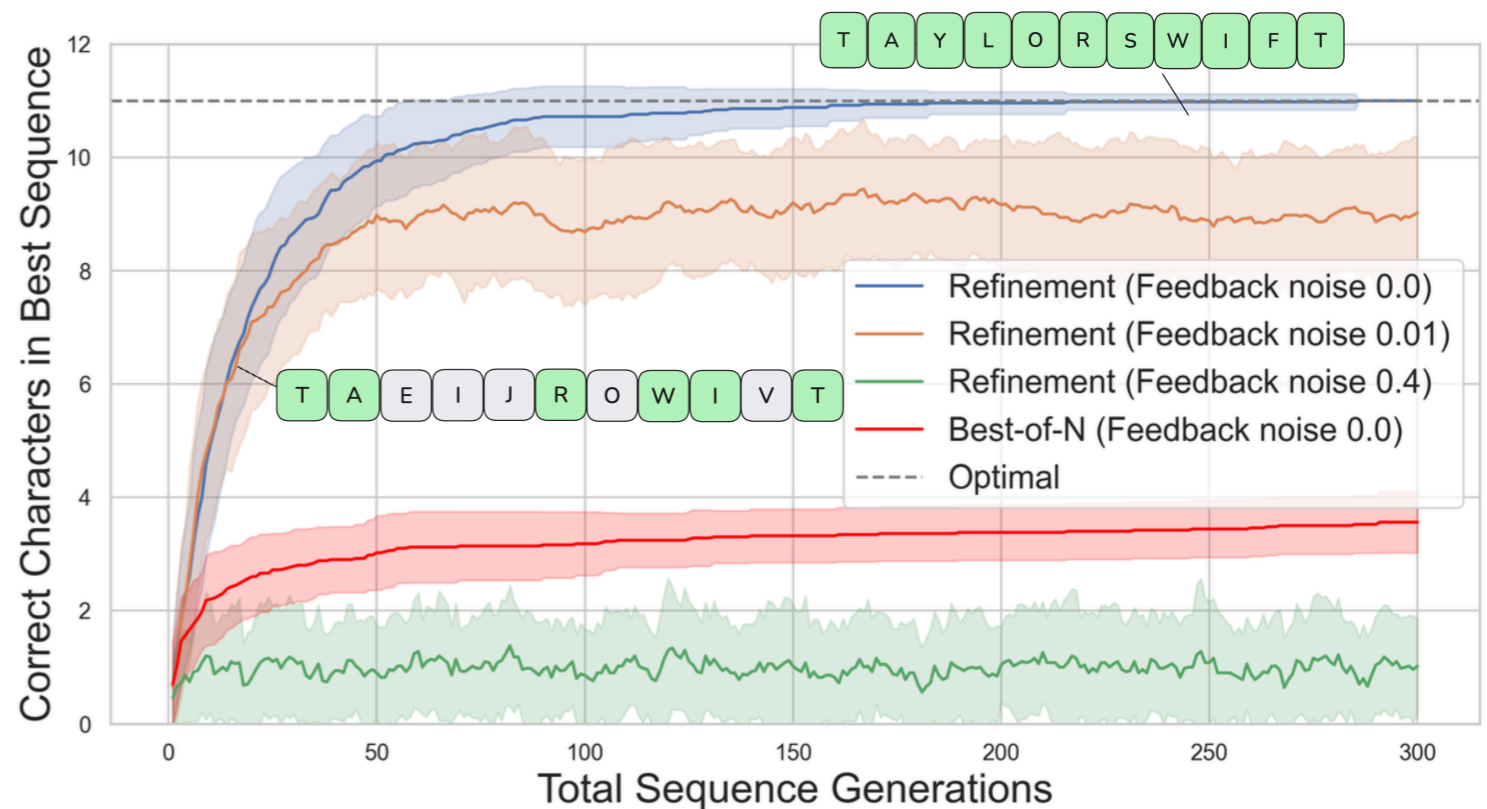
- *Self-refinement*: re-prompt the same model to generate, give feedback, and refine
- Mixed results:
  - Can work on tasks that are easy to evaluate
    - Example: we want 10 citations in the output and the model generation has 9 citations
  - Less clear for mathematical reasoning

# GSM8k (GPT 3.5)



# A toy model for refinement

- Task: generate “TAYLORSWIFT”
- Generator:
  - Simple unigram model,  $p(\text{character})$
- Feedback:
  - Which character positions are incorrect
  - + possibly add noise
- Refinement:
  - Regenerate only the incorrect positions



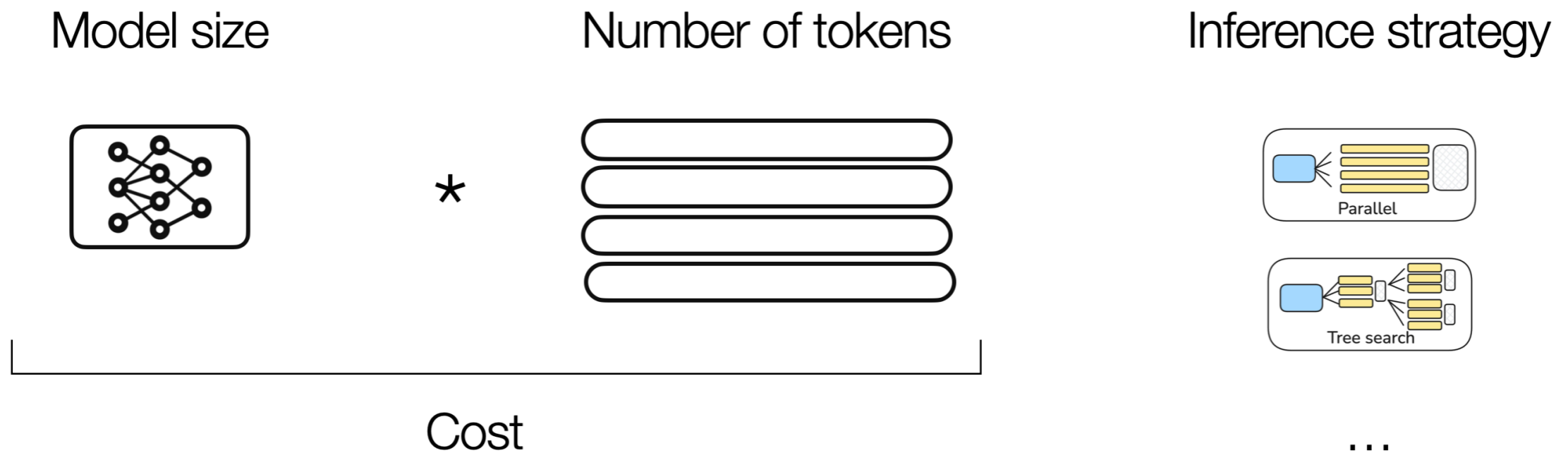
Intuition: refinement depends on how noisy the feedback is

# Meta-generation algorithms

- Strategies for calling a generator multiple times
- Common patterns
  - Parallel
    - Best-of-N, voting
  - Refinement
- **Next:** Inference scaling laws

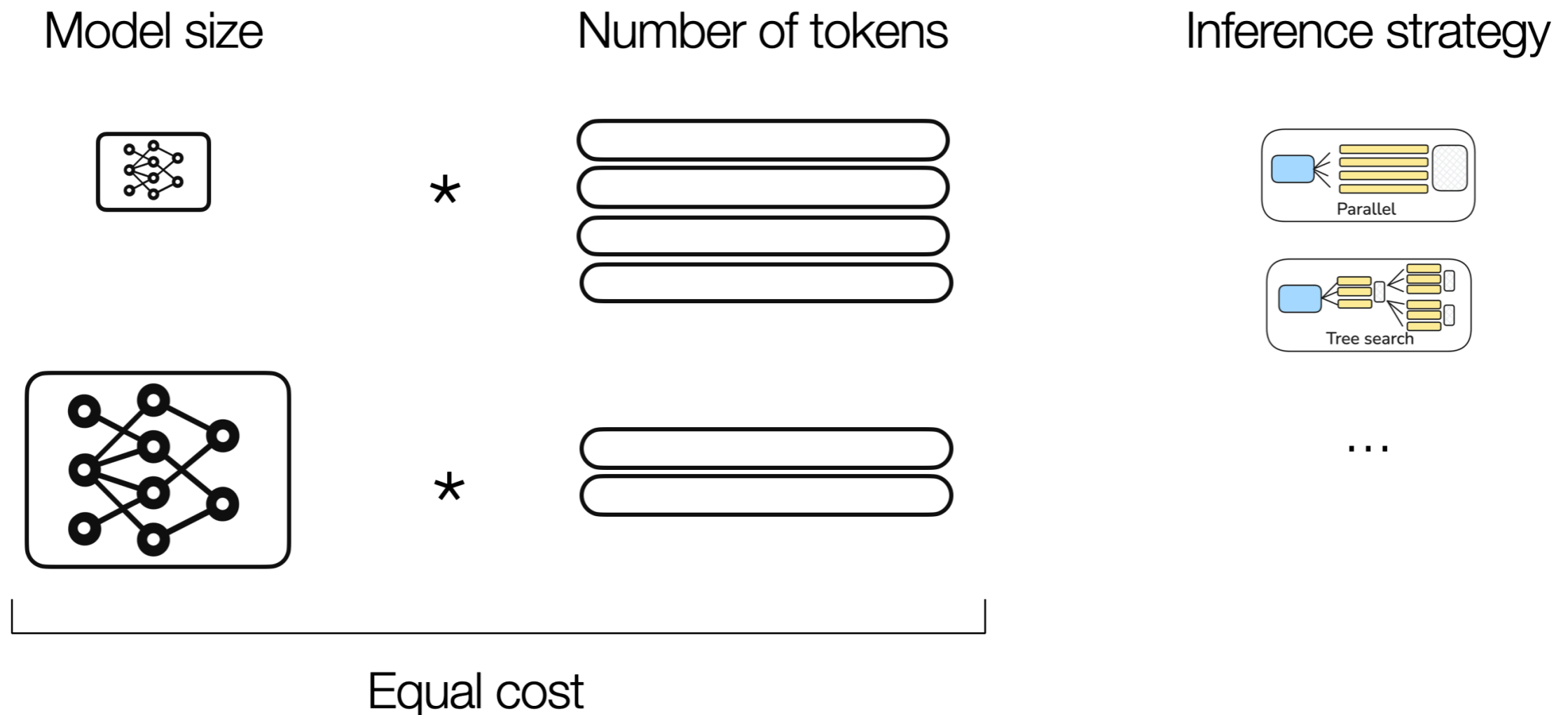
# Inference scaling laws

- Compute is a function of model size and number of generated tokens



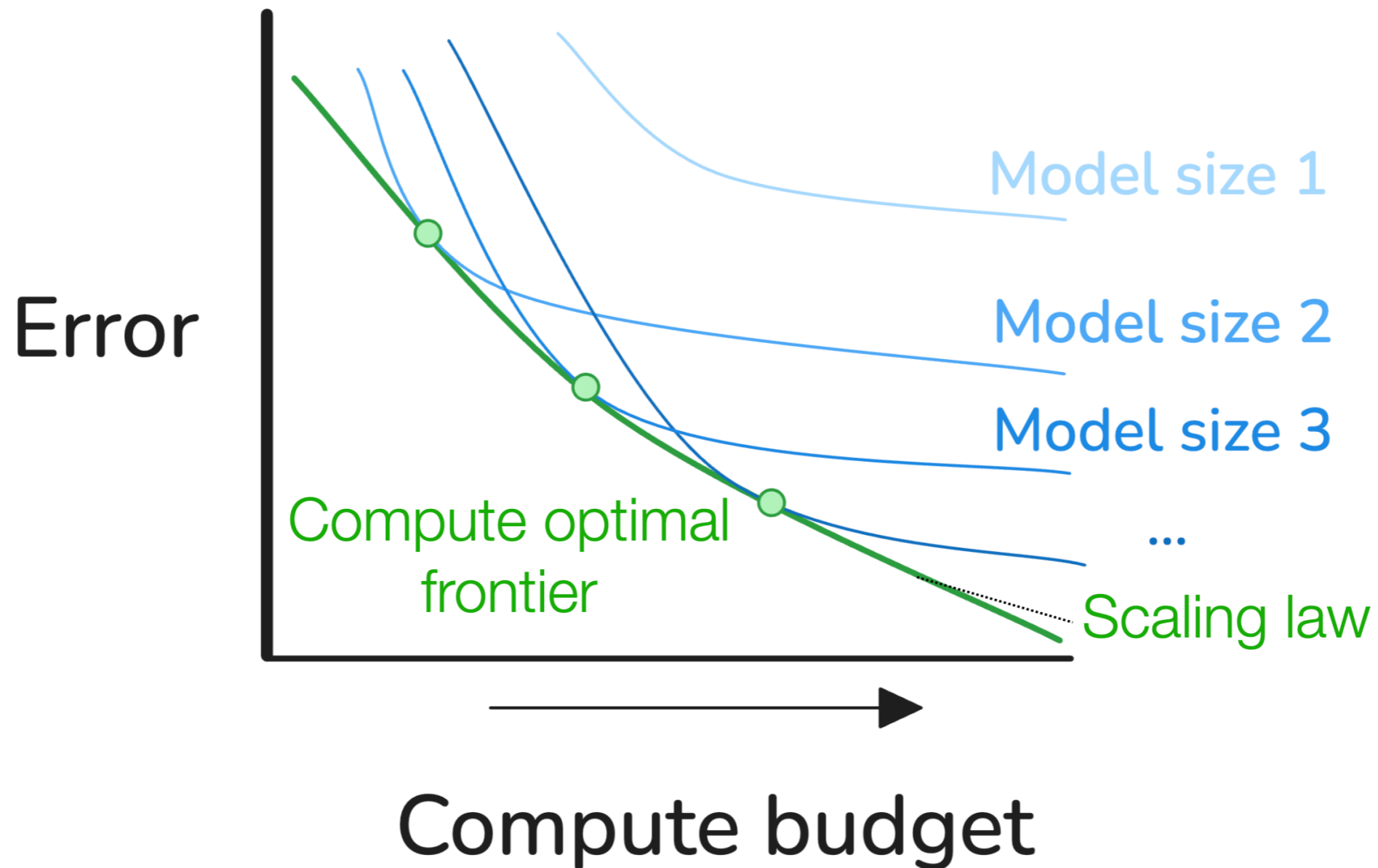
# Inference scaling laws

- We can increase model size and/or increase the number of tokens generated



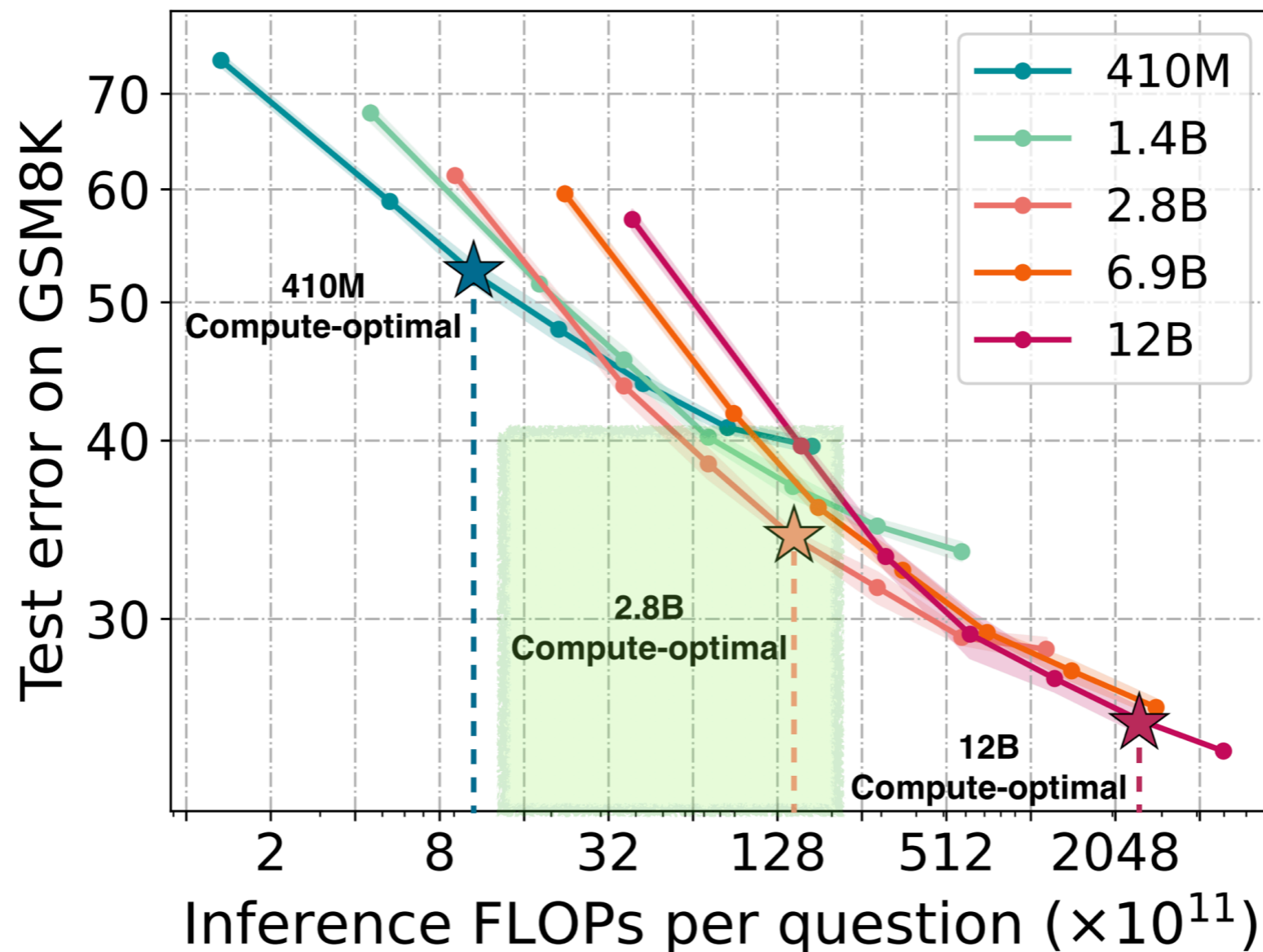
# Inference scaling laws

Fix strategy



# Inference scaling laws

- Using a smaller model and generating more can be preferred over using a larger model and generating less



# Today's lecture: advanced inference

- Test-time scaling strategies
  - Generating multiple times
  - **Generating longer outputs**

# Long chain-of-thought / reasoning models

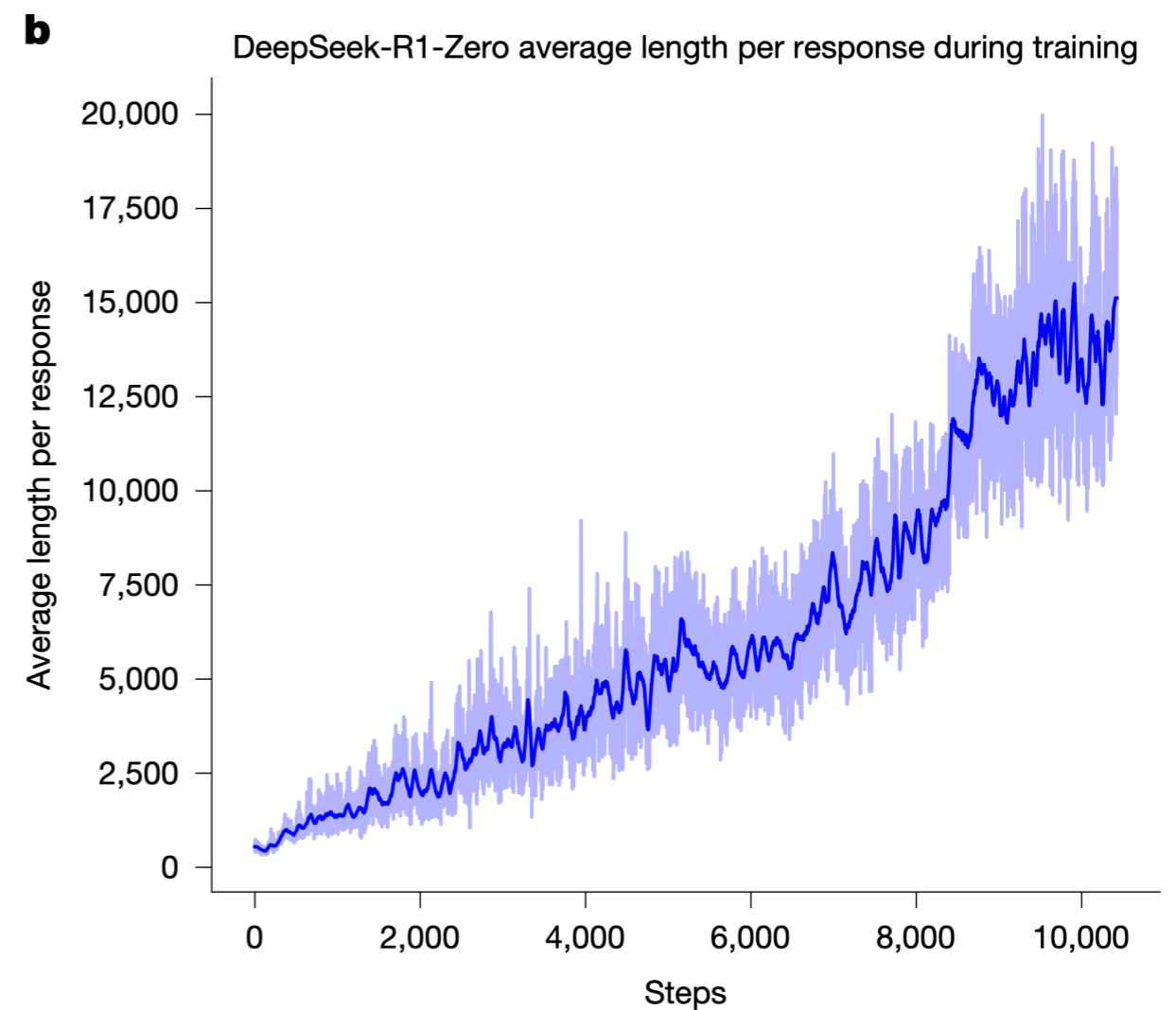
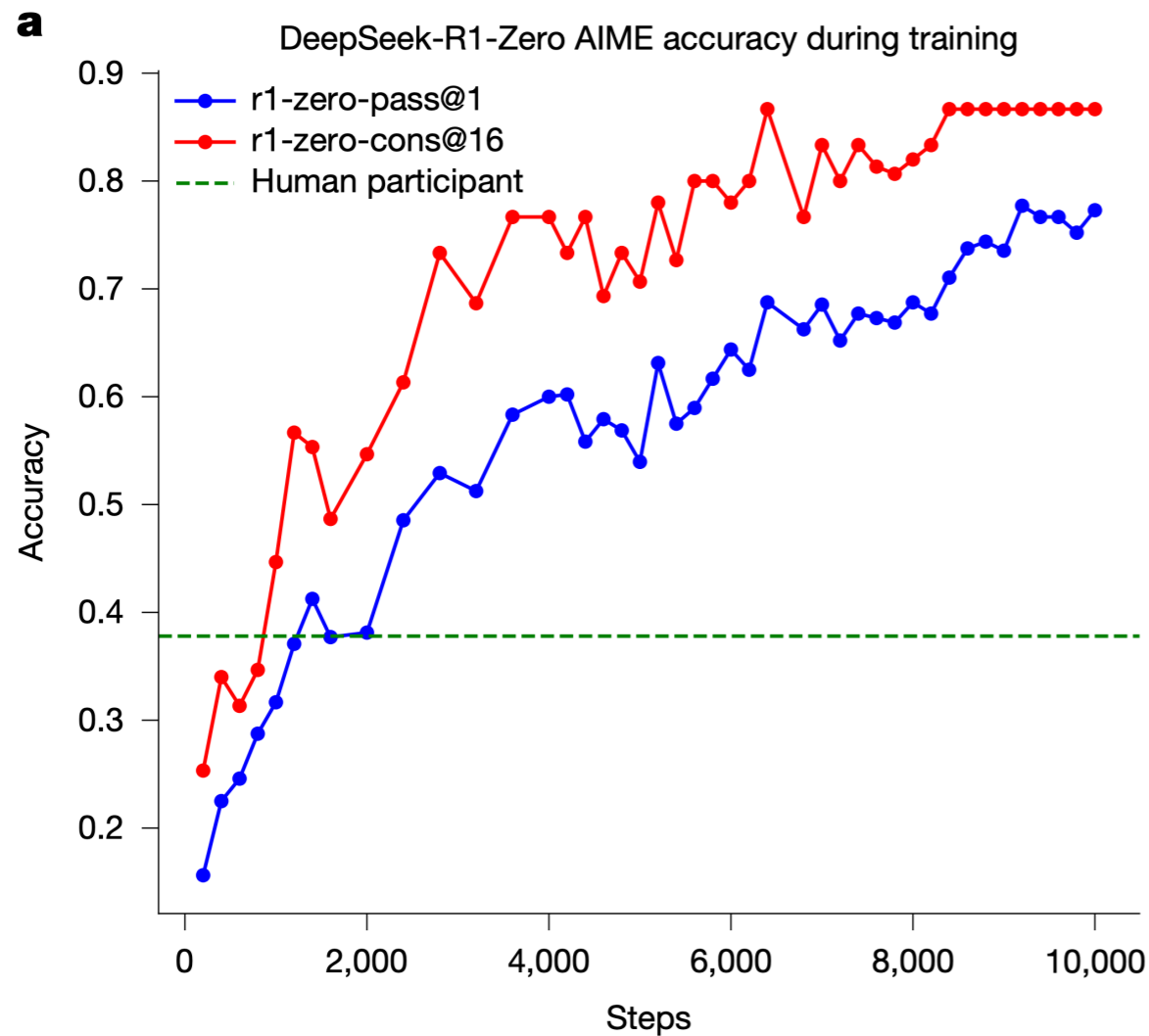
- Train a model to generate a “thought” prior to a final output

$$p_{\theta}(y, z | x)$$

- At inference time, just sample a thought+output
- In principle, the model can “learn a search algorithm”
  - Try alternatives, refine, etc. within the thought

# Long chain-of-thought / reasoning models

- Approach 1: reinforcement learning [see RL for LLMs lecture]



[DeepSeek 2025]

# Long chain-of-thought / reasoning models

---

Question: If  $a > 1$ , then the sum of the real solutions of  $\sqrt{a - \sqrt{a + x}} = x$  is equal to

---

Response: <think>

To solve the equation  $\sqrt{a - \sqrt{a + x}} = x$ , let's start by squaring both  $\dots$

$$\left(\sqrt{a - \sqrt{a + x}}\right)^2 = x^2 \implies a - \sqrt{a + x} = x^2.$$

Rearrange to isolate the inner square root term:

$$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$$

$\dots$

**Wait, wait. Wait. That's an aha moment I can flag here.**

Let's reevaluate this step-by-step to identify if the correct sum can be  $\dots$

We started with the equation:

$$\sqrt{a - \sqrt{a + x}} = x$$

First, let's square both sides:

$$a - \sqrt{a + x} = x^2 \implies \sqrt{a + x} = a - x^2$$

Next, I could square both sides again, treating the equation:  $\dots$

$\dots$

---

# Long chain-of-thought patterns

## 1. Uncertainty

- *Wait... / Hold on...*
- *Wait—actually, does this formula apply here?*

## 2. Branching, backtracking, retrying

- *Alternatively, generating functions could model this problem...*
- *Revisiting...*
- *Wait, I'm overthinking. Let's try again...*

## 3. Verification

- *Let's check if we made an error. We should verify...*
- *This is a contradiction, so we must have made a mistake.*
- *Let's test this with...*

## 4. Key Points

- *Key takeaway... / It's worth noting...*

## 5. Clarification

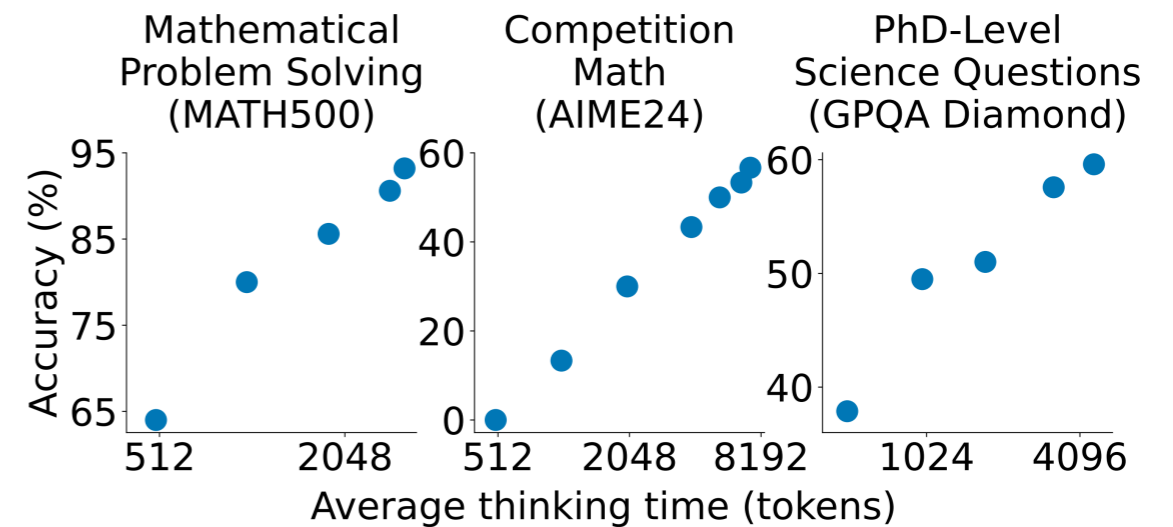
- *In other words... / To clarify...*

## 6. Synthesis

- *Ultimately... / Putting it all together...*

# Controlling the length: budget forcing

- Adhere to a length budget by forcing the model to generate “wait” or “final answer”
- Trade off tokens and performance



*Figure 1. Test-time scaling with s1-32B.* We benchmark s1-32B on reasoning-intensive tasks and vary test-time compute.

# Controlling the length: L1

**Example**

**Question**

Find the largest possible real part of  $[(75 + 117i)z + \frac{96+144i}{z}]$  where  $z$  is a complex number with  $|z| = 4$ . Let's think step by step and output the final answer within boxed{}. Think for 512 tokens.

**Correct Answer**

540

**Correctness**

✗ Incorrect

**Tokens Requested**

512

**Model Response**

Thinking Process

Okay, so I have this problem: I need to find the largest possible real part of the expression  $(75 + 117i)z + (96 + 144i)/z$ , where  $z$  is a complex number with  $|z| = 4$ . Hmm, let's break this down.

First, since  $|z| = 4$ ,  $z$  can be written as  $4e^{i\theta}$  for some angle  $\theta$ . That might help simplify things.

Let me denote  $z = 4e^{i\theta}$ . Then,  $1/z = (1/4)e^{-i\theta}$  because  $|z| = 4$ .

So, the expression becomes  $(75 + 117i) \cdot 4e^{i\theta} + (96 + 144i) \cdot (1/4)e^{-i\theta}$ .

Simplifying, that's  $4(75 + 117i)e^{i\theta} + (24 + 36i)e^{-i\theta}$ .

Now, I need to find the real part of this expression. Let's denote  $\text{Re}[(75 + 117i)z + (96 + 144i)/z]$ .

Breaking it down,  $\text{Re}[(75 + 117i)z] + \text{Re}[(96 + 144i)/z]$ .

Since  $z = 4e^{i\theta}$ ,  $\text{Re}[(75 + 117i) \cdot 4e^{i\theta}] + \text{Re}[(96 + 144i) \cdot (1/4)e^{-i\theta}]$ .

This becomes  $4\text{Re}[(75 + 117i)e^{i\theta}] + (1/4)\text{Re}[(96 + 144i)e^{-i\theta}]$ .

Now, let's compute  $\text{Re}[(75 + 117i)e^{i\theta}]$ . This is  $75 \cos \theta + 117 \sin \theta$ .

Similarly,  $\text{Re}[(96 + 144i)e^{-i\theta}]$  is  $96 \cos \theta - 144 \sin \theta$ .

So, putting it all together, the real part is  $4(75 \cos \theta + 117 \sin \theta) + (1/4)(96 \cos \theta - 144 \sin \theta)$ .

Simplifying, that's  $300 \cos \theta + 468 \sin \theta + 24 \cos \theta - 36 \sin \theta$ .

Combining like terms, we get  $(300 + 24) \cos \theta + (468 - 36) \sin \theta = 324 \cos \theta + 432 \sin \theta$ .

Now, we need to find the maximum of  $324 \cos \theta + 432 \sin \theta$ . This can be written as  $R \cos(\theta - \phi)$ , where  $R = \sqrt{324^2 + 432^2}$ .

Calculating  $R$ :  $324^2 = 104,976$ ,  $432^2 = 186,624$ . Sum is  $391,600$ .  $R = \sqrt{391,600} \approx 625.6$ .

Thus, the maximum real part is approximately 625.6.

The largest possible real part is 625.6.

Incorrect (512 tokens)

**Example**

**Question**

Find the largest possible real part of  $[(75 + 117i)z + \frac{96+144i}{z}]$  where  $z$  is a complex number with  $|z| = 4$ . Let's think step by step and output the final answer within boxed{}. Think for 3600 tokens.

**Correct Answer**

540

**Correctness**

✓ Correct

**Tokens Requested**

3600

**Model Response**

Thinking Process (not shown to user)

Okay, so I have this problem: I need to find the largest possible real part of the expression  $(75 + 117i)z + (96 + 144i)/z$ , where  $z$  is a complex number with  $|z| = 4$ . Hmm, that seems a bit involved, but let's break it down step by step.

First, let me understand what is given. We have a complex number  $z$  with magnitude 4, so  $z$  can be written in the form  $z = 4e^{i\theta}$ , where  $\theta$  is the argument of  $z$ . That might be useful because it allows us to express  $z$  in terms of its angle, which could simplify the expression.

The expression we need to consider is  $(75 + 117i)z + \frac{96+144i}{z}$ . We are asked to find the largest possible real part of this expression. So, let's denote the expression as  $E = (75 + 117i)z + \frac{96+144i}{z}$ .

[Reasoning process truncated for brevity ...]

Then, adding  $104,976 + 186,624 = 291,600$ .  $\sqrt{291,600} = \sqrt{2916 \cdot 100} = \sqrt{2916} \cdot 10$ . Now,  $\sqrt{2916}$ : let's see,  $54^2 = 2916$ , because  $50^2 = 2500$ ,  $54^2 = 2916$ . So,  $\sqrt{2916} = 54$ , thus  $\sqrt{2916} \cdot 100 = 54 \cdot 100 = 5400$ . So, that seems correct.

Therefore, the maximum value of  $\text{Re}(E)$  is 540.

Therefore, the largest possible real part is 540.

Given the expression  $(75 + 117i)z + \frac{96+144i}{z}$  where  $z$  is a complex number with  $|z| = 4$ , we need to find the largest possible real part of this expression.

First, we express  $z$  as  $4e^{i\theta}$  since  $|z| = 4$ . Then, we substitute  $z$  and  $\frac{1}{z}$  into the expression:

$$(75 + 117i)z + \frac{96 + 144i}{z}$$

[Output truncated for brevity ...]

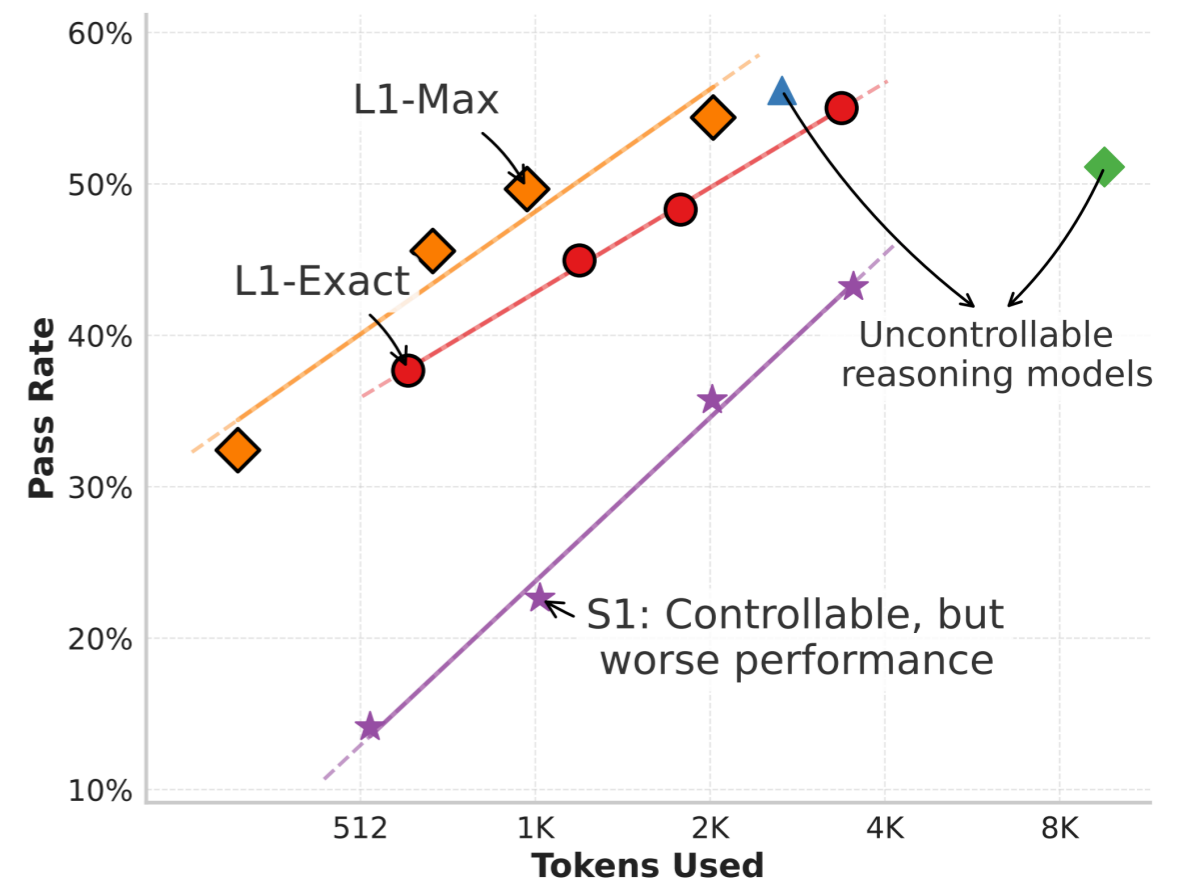
Thus, the largest possible real part is:

540

Correct (3600 tokens)

# Controlling the length: L1

- Train model with RL to adhere to length constraints
- E.g. “use up to 2000 tokens” provided in the prompt
- Reward: correctness and length constraint penalty



# Today's lecture: advanced inference

- Test-time scaling strategies
  - Generating multiple times
    - Parallel sampling
    - Voting
    - Inference scaling laws
  - Generating longer outputs
    - Reasoning models / long CoT
    - Controlling the reasoning length

Thank you