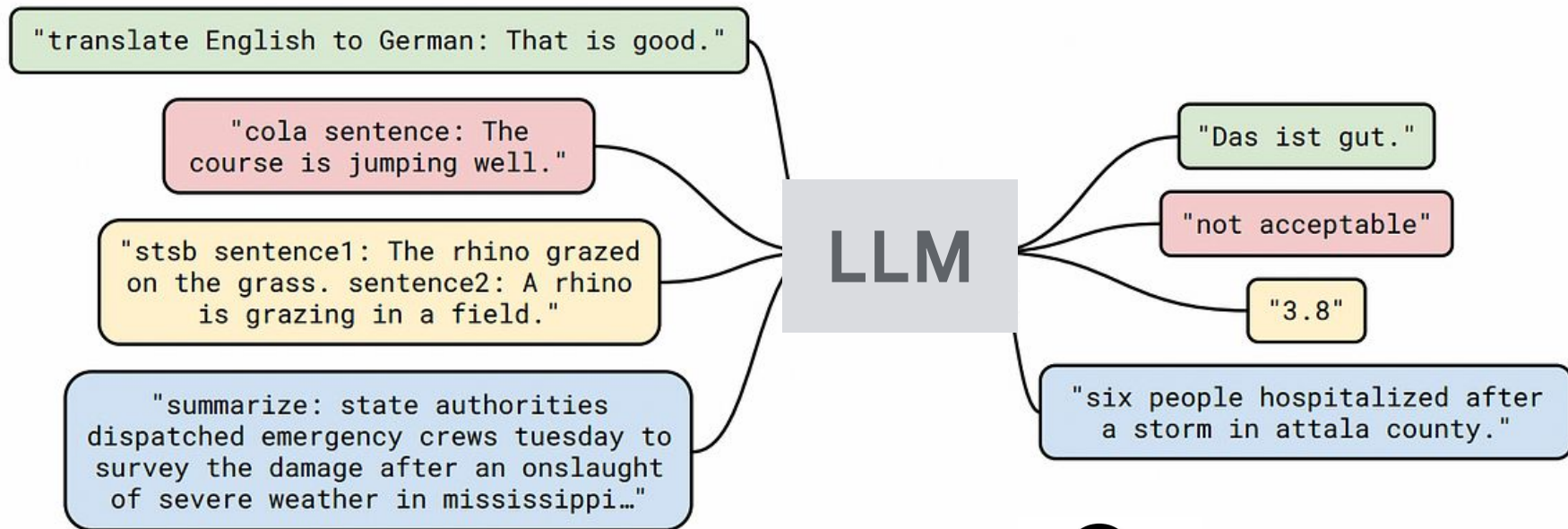


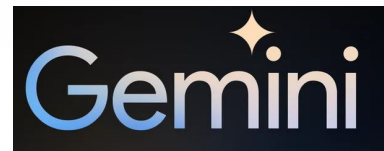
IMPROVING LLM REASONING USING SELF-GENERATED DATA: RL AND VERIFIERS

Rishabh Agarwal
Research Scientist, Google DeepMind

Large Language Models (LLMs)



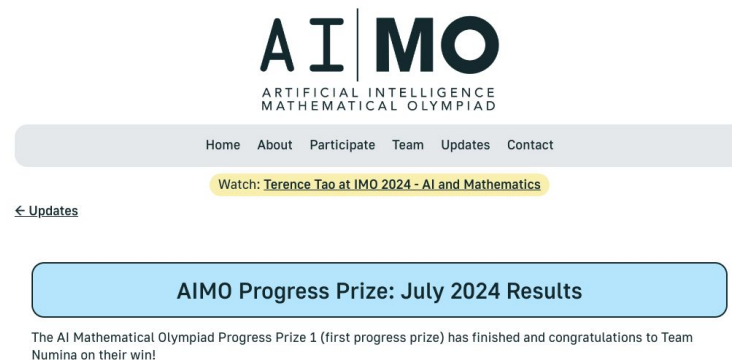
GPT - 4



Learning to Reason with LLMs

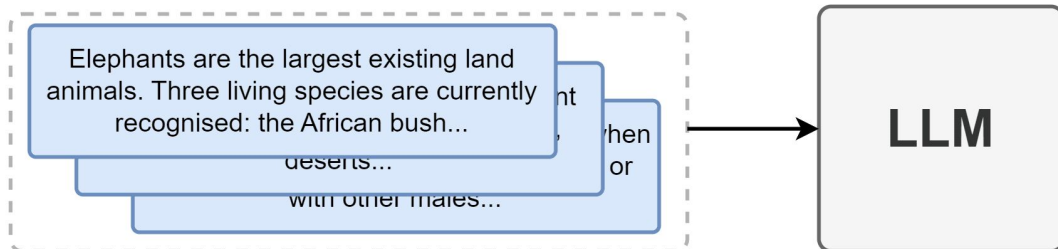
We are introducing OpenAI o1, a new large language model trained with reinforcement learning to perform complex reasoning. o1 thinks before it answers—it can produce a long internal chain of thought before responding to the user.

AI achieves silver-medal standard solving International Mathematical Olympiad problems



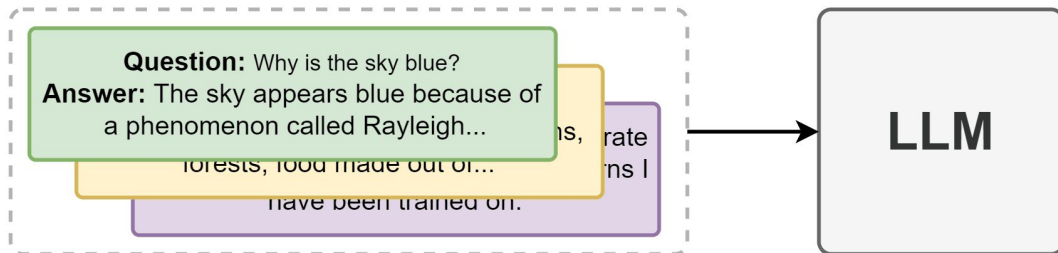
Training LLMs needs high quality data

Stage 1: Pretraining



Arbitrary Unstructured Data

Stage 2: Instruction Tuning



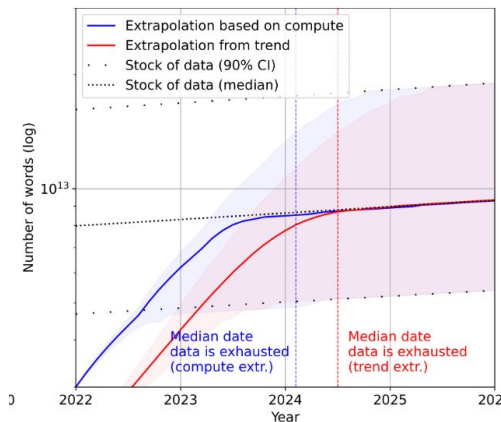
Task-Related Data
(Sample interactions, RLHF, etc.)

High quality
data,
scraped
from web or
collected
from
humans.

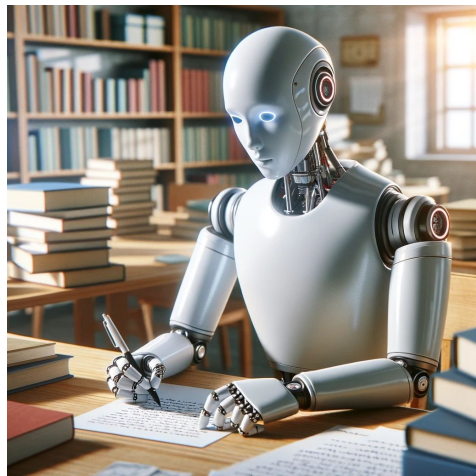
Are we running out of high-quality data?

Will We Run Out of ML Data? Evidence From Projecting Dataset Size Trends

- Time-consuming and expensive to scale
- Hard to create for complex tasks

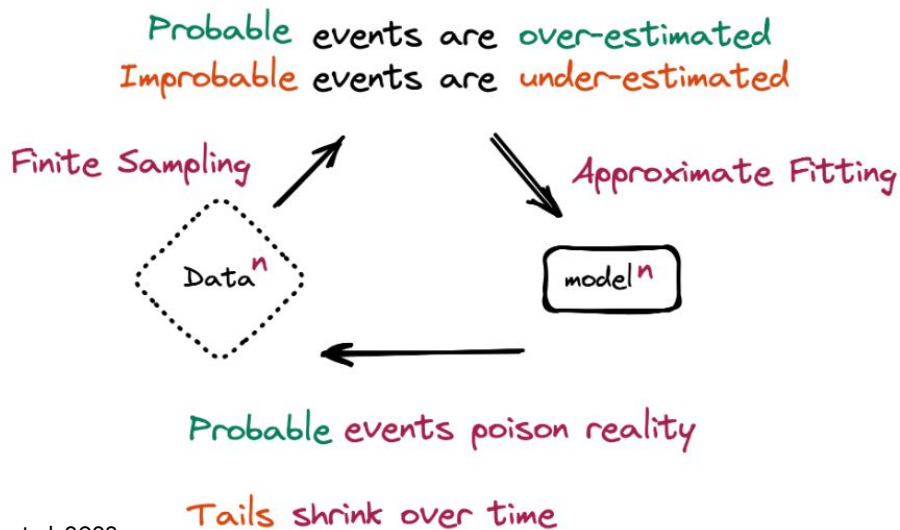


Synthetic data to the rescue?



What if the models could generate their own training outputs?

**Naively
doing so
can
result in
model
collapse!**



Synthetic data to the rescue?

Verification can often be **easier** than Generation!

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

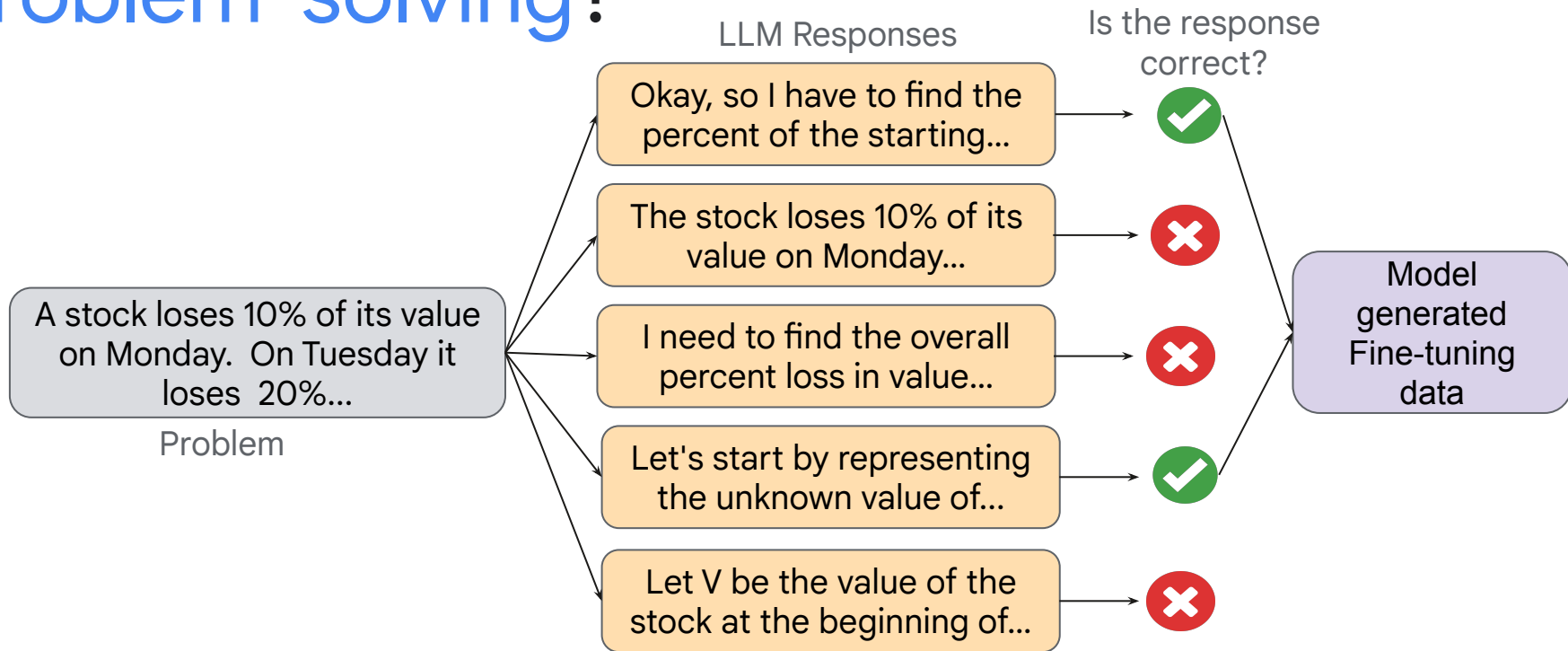
Solving sudoku puzzles is
harder than checking one!

GIVEN A STRING, FIND THE LENGTH OF
THE LONGEST SUBSTRING WITHOUT
REPEATING CHARACTERS.

Generating code can be harder than verifying it
via test case execution.

Can we use model-generated data for training
given access to some form of **feedback**?

How do we self-generate data for problem-solving?



A simple recipe for self-training (ReST^{EM})

Repeat this process a few times:

1. Generate samples from the model and filter them using binary feedback. (E-step)
2. Fine-tune the model on these samples (M-step)

This process corresponds to **expectation-maximization based RL**! Check the math in the paper.

Expectation-Maximization explained

$$\log p(O = 1 | \mathbf{x}) := \log \sum_{\mathbf{y}} p_{\theta}(\mathbf{y} | \mathbf{x}) p(O = 1 | \mathbf{x}, \mathbf{y})$$

$$\begin{aligned} \log p(O = 1 | \mathbf{x}) &= \log \mathbb{E}_{q(\mathbf{y} | \mathbf{x})} \left[\frac{p(O = 1 | \mathbf{x}, \mathbf{y}) p_{\theta}(\mathbf{y} | \mathbf{x})}{q(\mathbf{y} | \mathbf{x})} \right] \\ &\geq \mathbb{E}_{q(\mathbf{y} | \mathbf{x})} \left[\log \frac{p(O = 1 | \mathbf{x}, \mathbf{y}) p_{\theta}(\mathbf{y} | \mathbf{x})}{q(\mathbf{y} | \mathbf{x})} \right] \quad (\text{Jensen's inequality}) \\ &= \mathbb{E}_{q(\mathbf{y} | \mathbf{x})} [\log p(O = 1 | \mathbf{x}, \mathbf{y})] - \text{KL} [q(\mathbf{y} | \mathbf{x}) || p_{\theta}(\mathbf{y} | \mathbf{x})] \\ &=: L(p_{\theta}, q) \end{aligned}$$

Problem-Solving tasks: Math & Coding

Hendrycks MATH

Problem: The equation $x^2 + 2x = i$ has two complex solutions. Determine the product of their real parts.

Solution: Complete the square by adding 1 to each side.

Then $(x + 1)^2 = 1 + i = e^{\frac{i\pi}{4}} \sqrt{2}$, so $x + 1 = \pm e^{\frac{i\pi}{8}} \sqrt[4]{2}$.

The desired product is then

$$\begin{aligned} & (-1 + \cos(\frac{\pi}{8}) \sqrt[4]{2}) (-1 - \cos(\frac{\pi}{8}) \sqrt[4]{2}) = \\ & 1 - \cos^2(\frac{\pi}{8}) \sqrt{2} = 1 - \frac{(1 + \cos(\frac{\pi}{4}))}{2} \sqrt{2} = \boxed{\frac{1 - \sqrt{2}}{2}}. \end{aligned}$$

APPS Coding (Intro)

We will buy a product for N yen (the currency of Japan) at a shop. If we use only 1000-yen bills to pay the price, how much change will we receive? Assume we use the minimum number of bills required.

-----Constraints----- - $1 \leq N \leq 10000$ - N is an integer.

-----Input----- Input is given from Standard Input in the following format: N

-----Output----- Print the amount of change as an integer.

-----Sample Input-----

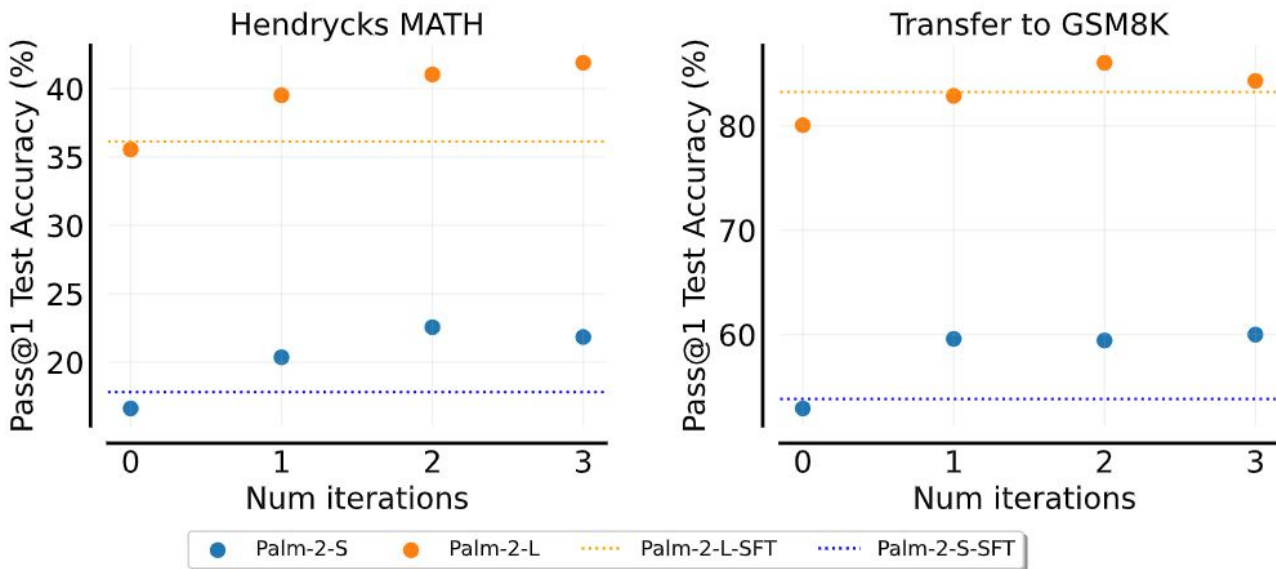
1900

-----Sample Output-----

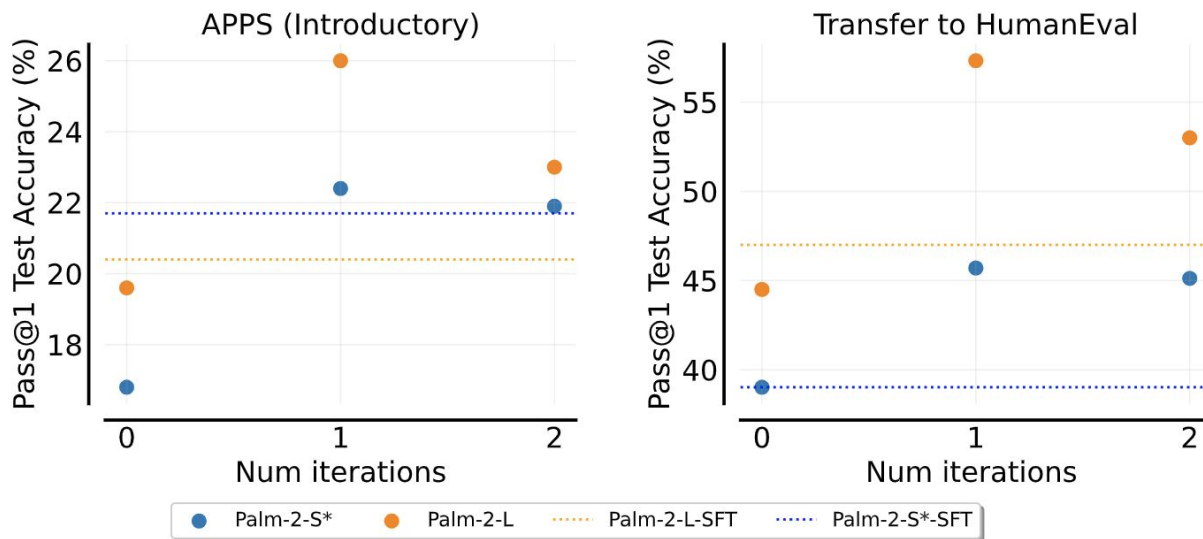
100

We will use two 1000-yen bills to pay the price and receive 100 yen in change.

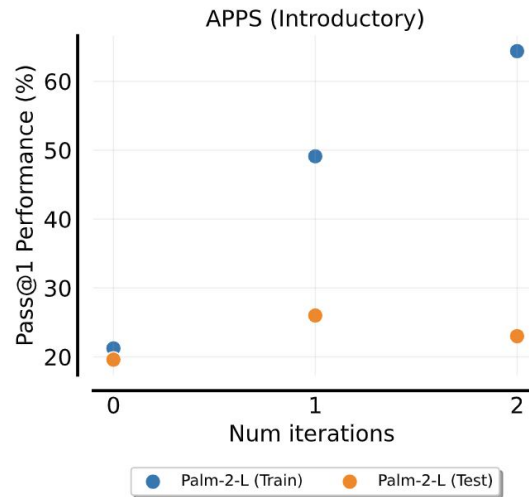
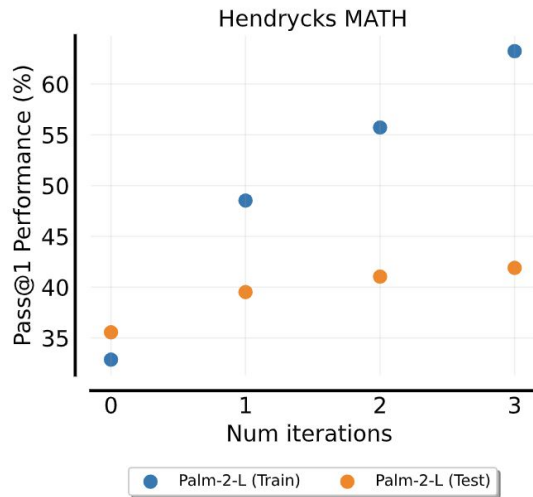
This... beats human data!



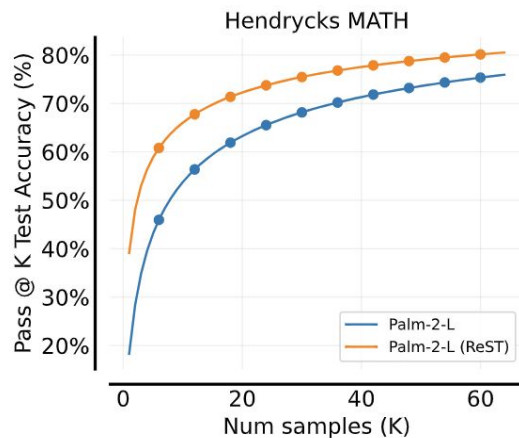
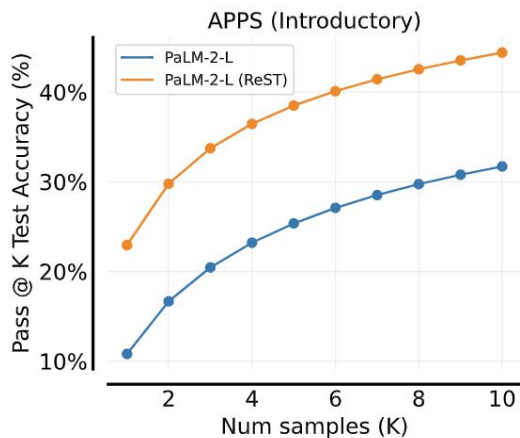
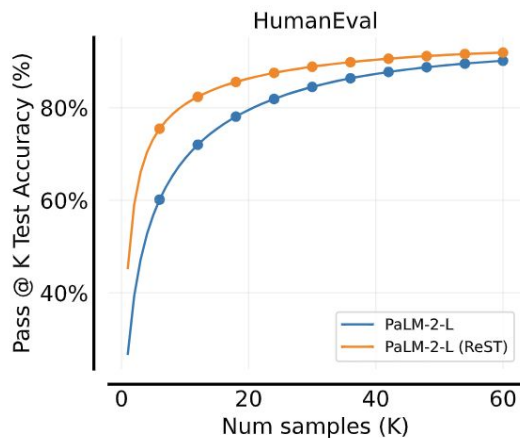
ReST^{EM} works on coding too.



Overfitting is an issue

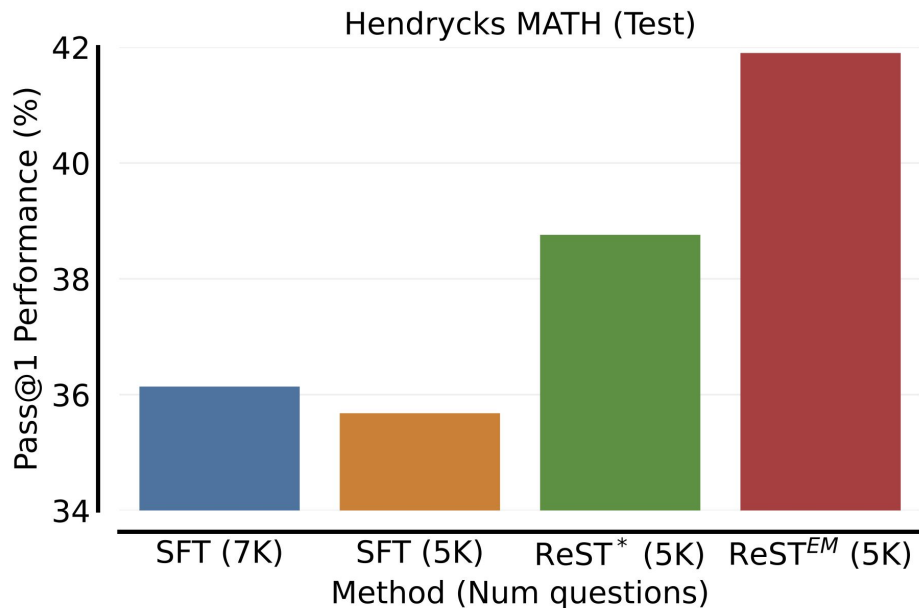


Pass@K performance improves as well

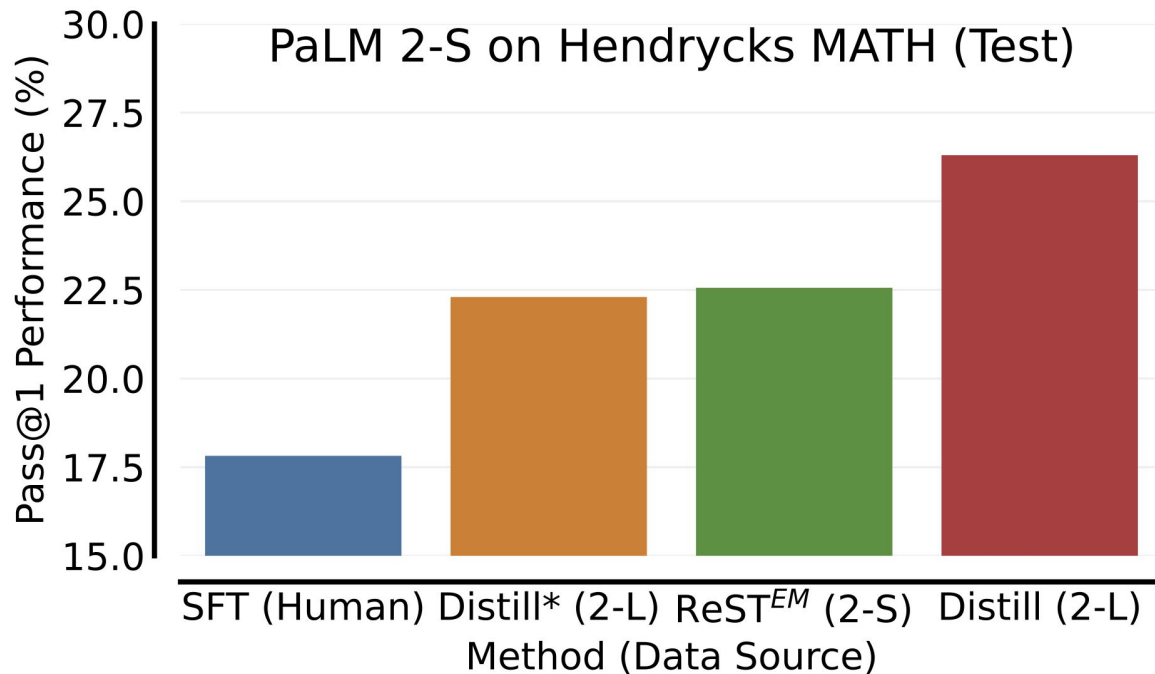


Pass@K measures the probability that at least one of the top k-generated solution for a problem is correct.

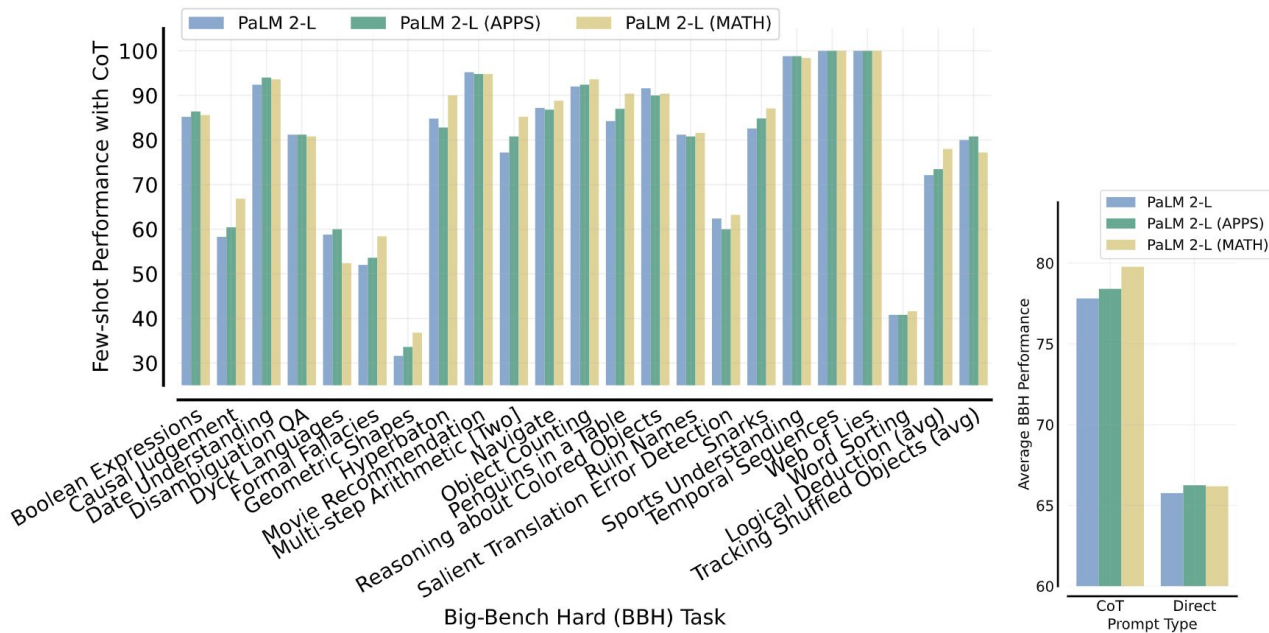
Apples-to-Apples Comparison



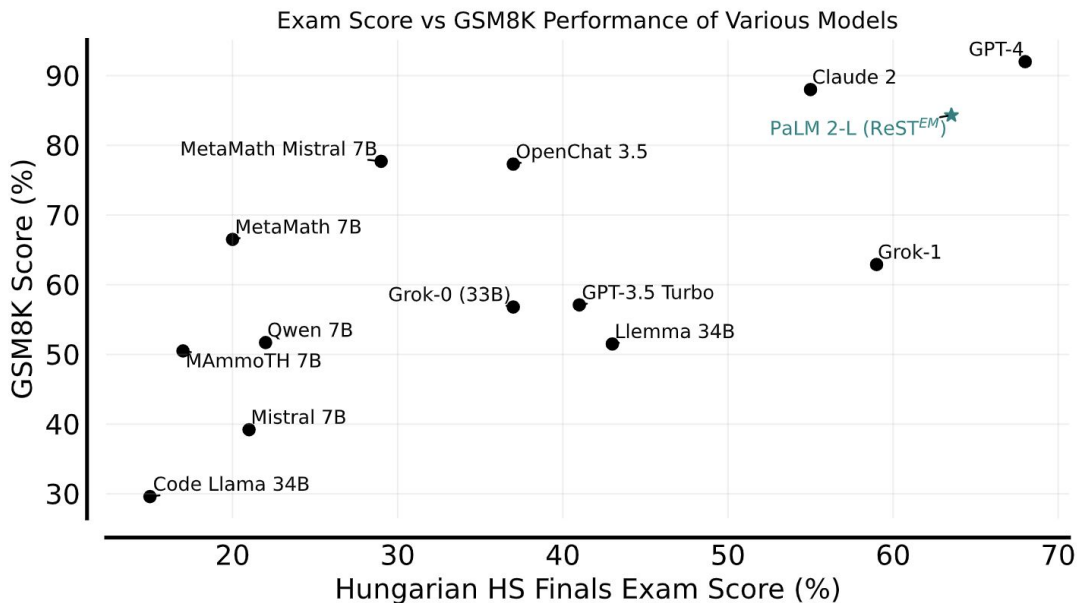
Distilling Palm-2-S using L



Impact on reasoning tasks



Held-Out Eval: 2023 Hungarian HS Exam



Things we learned so far:

- Self-generated data improves performance, given reliable reward.
- Self-generated data can often outperform human data – it's more in-distribution!

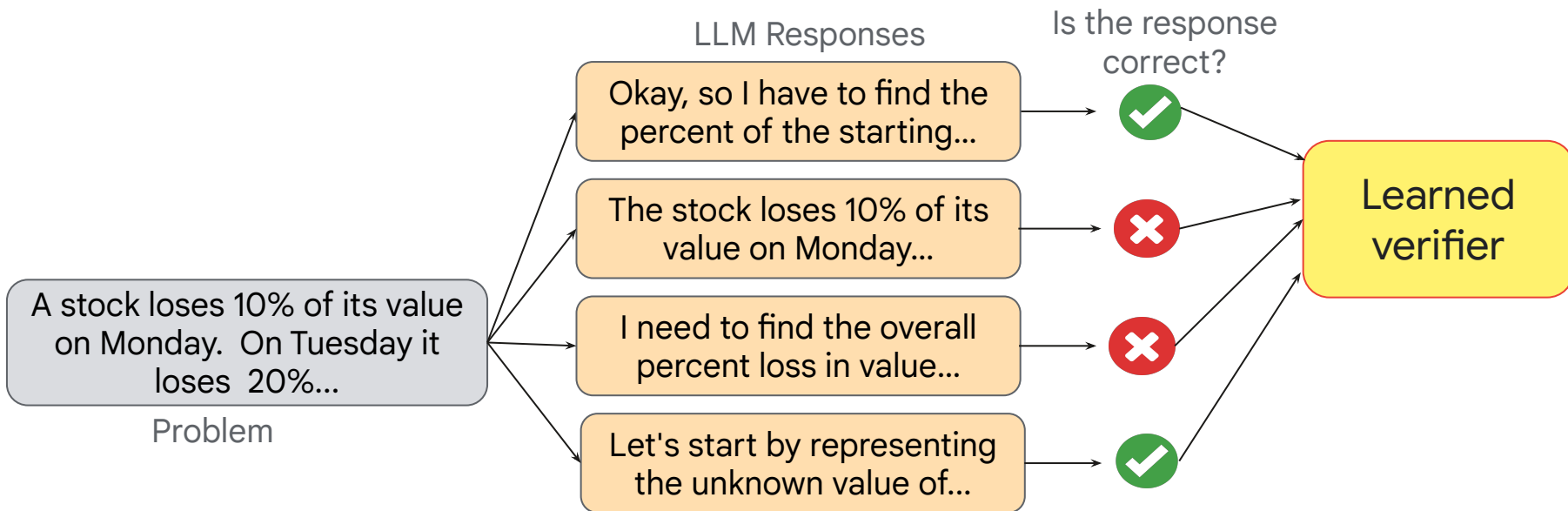
Revisiting ReST^{EM}

Repeat this process a few times:

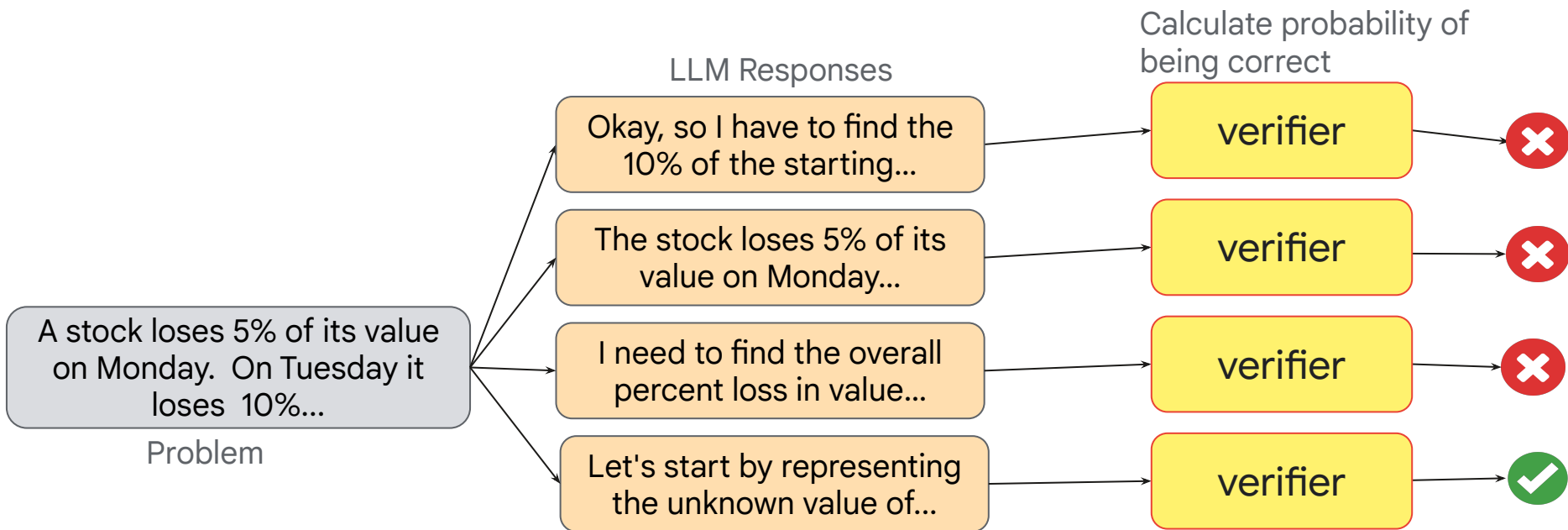
1. Generate samples from the model and **filter them using binary feedback.**
2. Fine-tune the model on these samples

Discard the large amounts of incorrect solutions generated during this process, potentially neglecting valuable information!

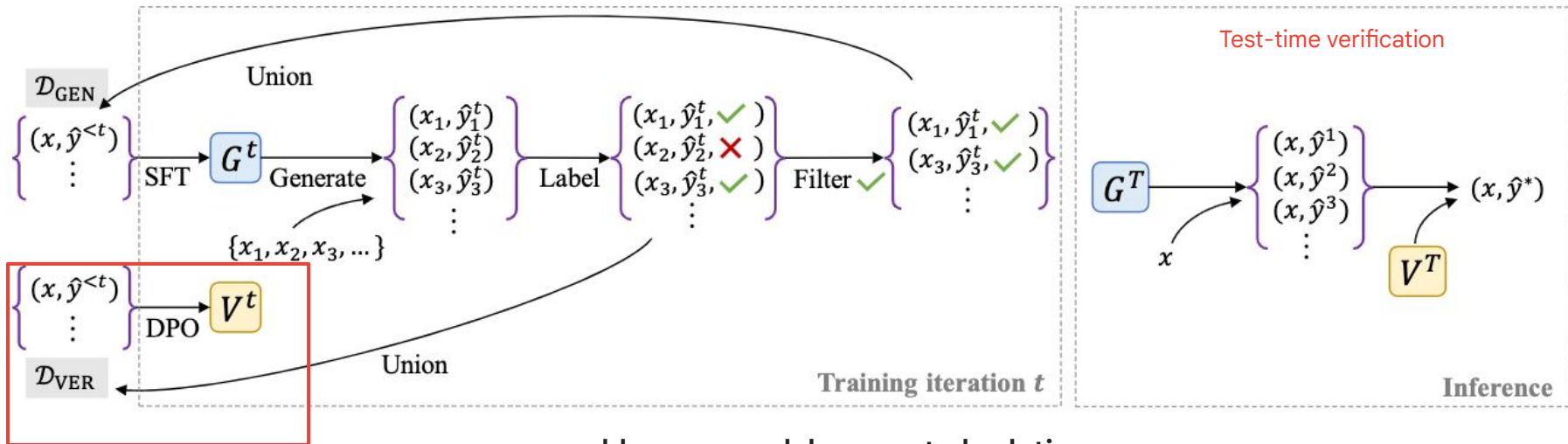
Incorrect solutions for training **verifiers**



How to use a **verifier**?

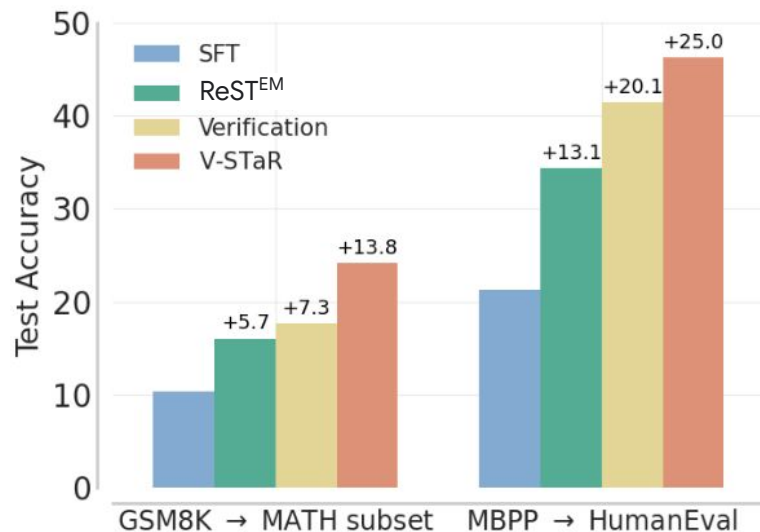
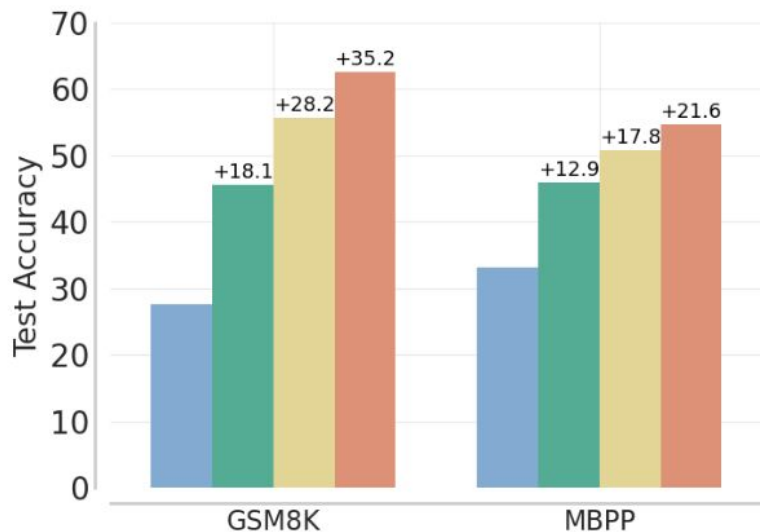


Idea: Augmenting ReST^{EM} with a verifier



x = problem, y = model-generated solution

V-STaR: ReST^{EM} + verifier works quite well!

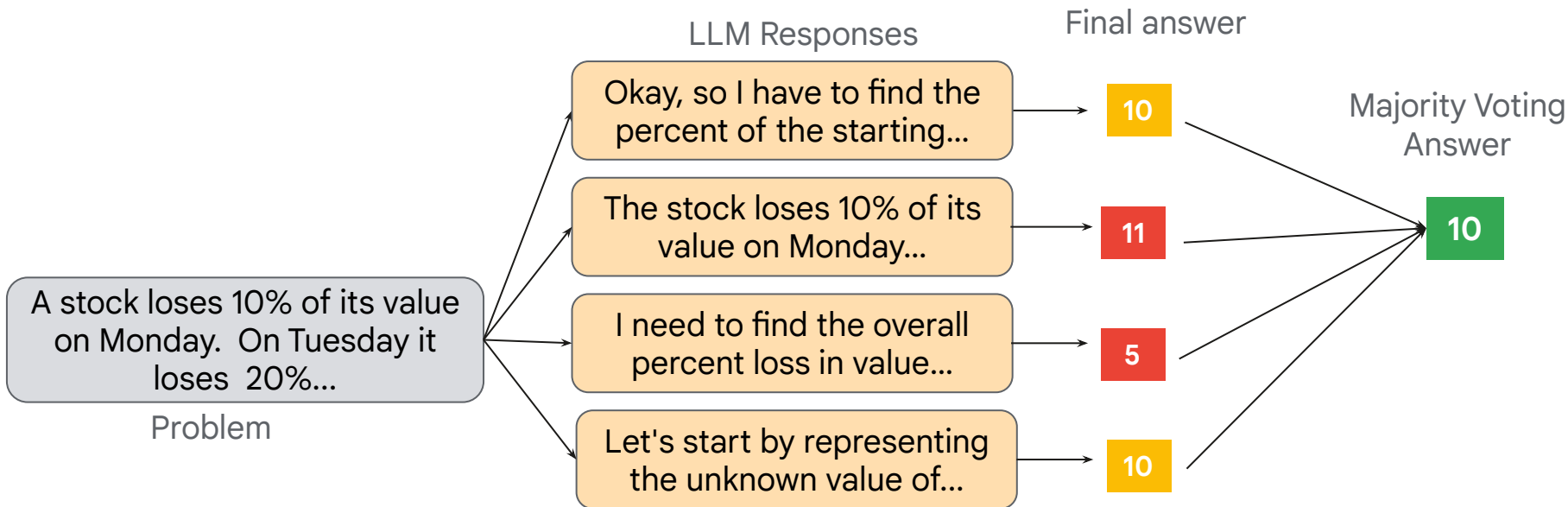


Large gains on math and code reasoning with LLaMA2 7B and 13B models.

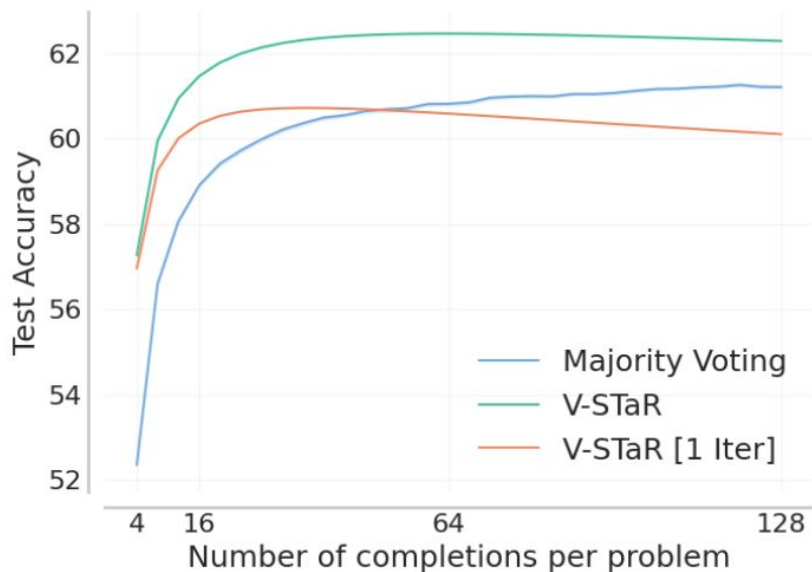
V-STaR: Performance across iterations



A Strong Baseline: Majority Voting



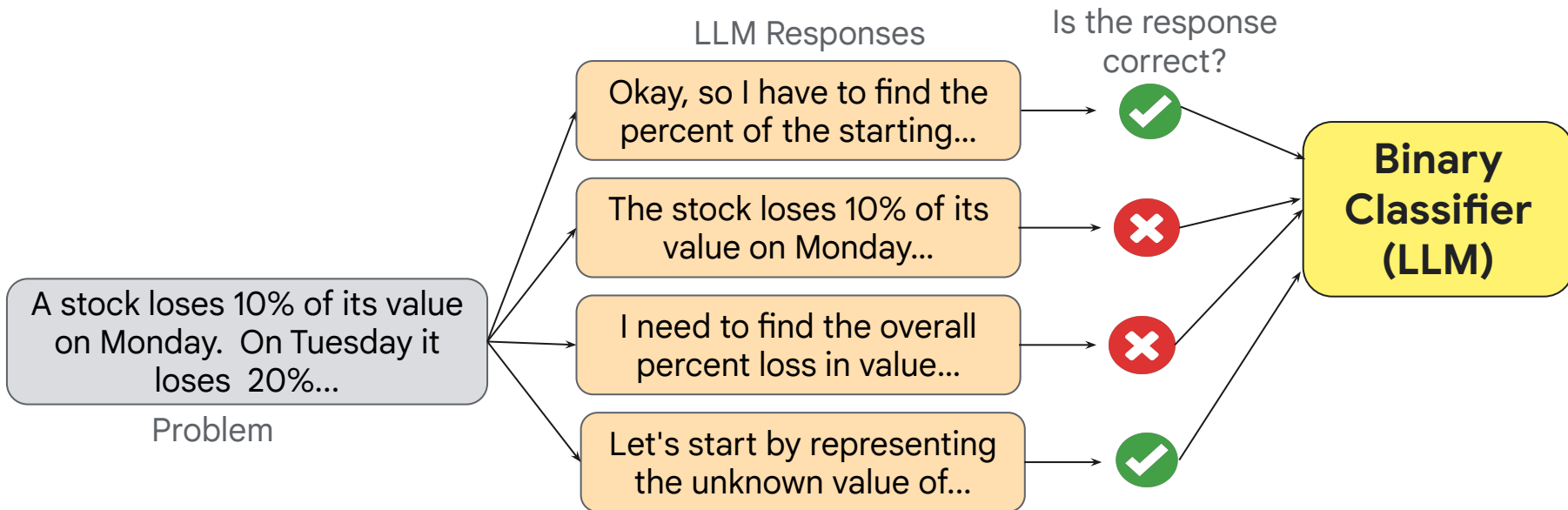
V-STaR Scales Better With Test-Time Compute



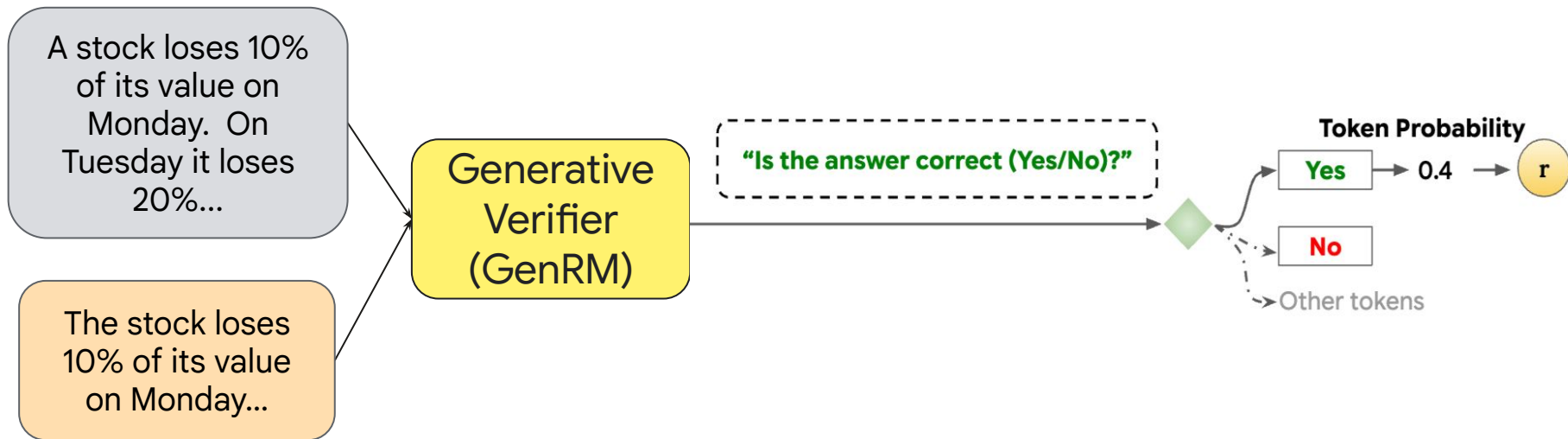
Things we learned so far:

- Self-generated data improves performance, given reliable reward.
- Self-generated data can often outperform human data – it's more in-distribution!
- We can train a verifier, using both correct and incorrect solutions.
- Verifiers can help make use of test-time compute.

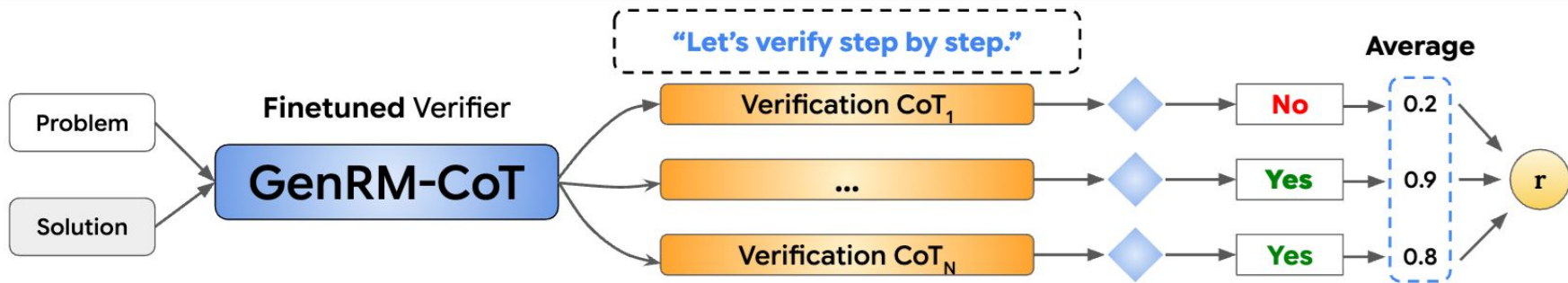
Revisiting Verifiers



Train Verifiers as Next-token Predictors



Generative Verifiers Can Use Chain-of-Thought (CoT) Reasoning



Generative Verifiers Can Use Chain-of-Thought Reasoning

Problem: Tim decides to cancel his cable subscription and get streaming services. He gets Netflix for \$10 a month. Hulu and Disney Plus normally cost \$10 a month each but he saves 20% for bundling. How much money does he save by cancelling his \$60 cable package?

Solution: Tim pays \$60 for cable. He gets Netflix for 10 and **the bundle of Hulu and Disney Plus costs $\$10 * 80\% = \8** . So he pays $\$10 + \$8 = \$18$ for Netflix and the bundle. Now he saves $\$60 - \$18 = \$42$. The answer is 42.

Discriminative RM correctness score: **0.999**

GenRM-CoT: Let's verify step by step.

...

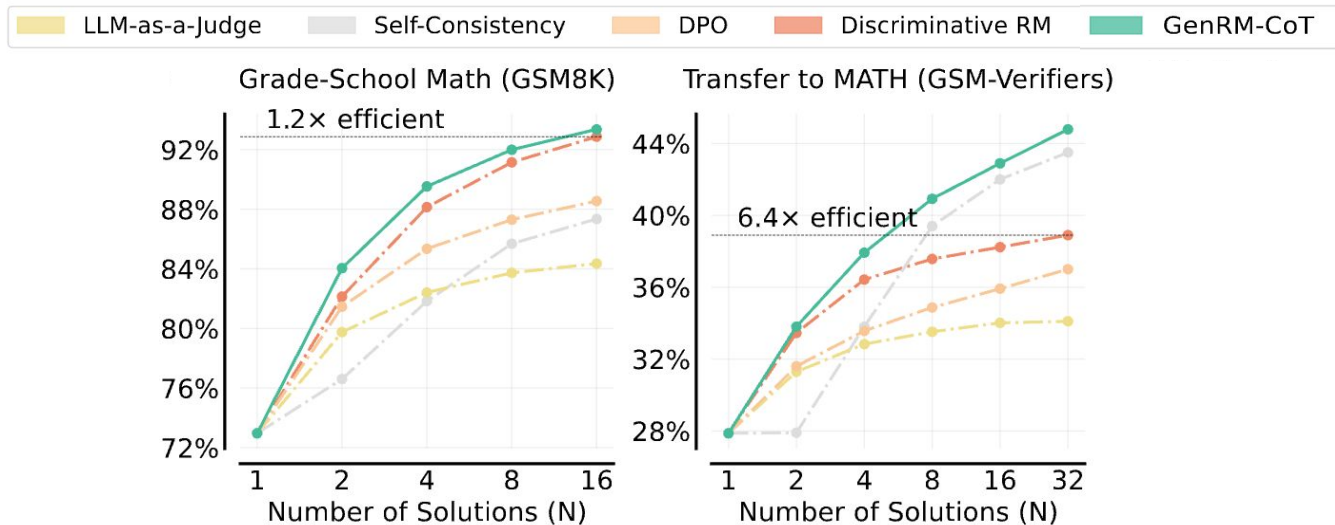
Step 3. ****The bundle of Hulu and Disney Plus costs $\$10 * 80\% = \8 .** (Incorrect)**

- Expected Answer: **The bundle of Hulu and Disney Plus costs $\$10 + \$10 = \$20$. With the 20% discount, the total cost is $\$20 * 0.8 = \16 .** ...

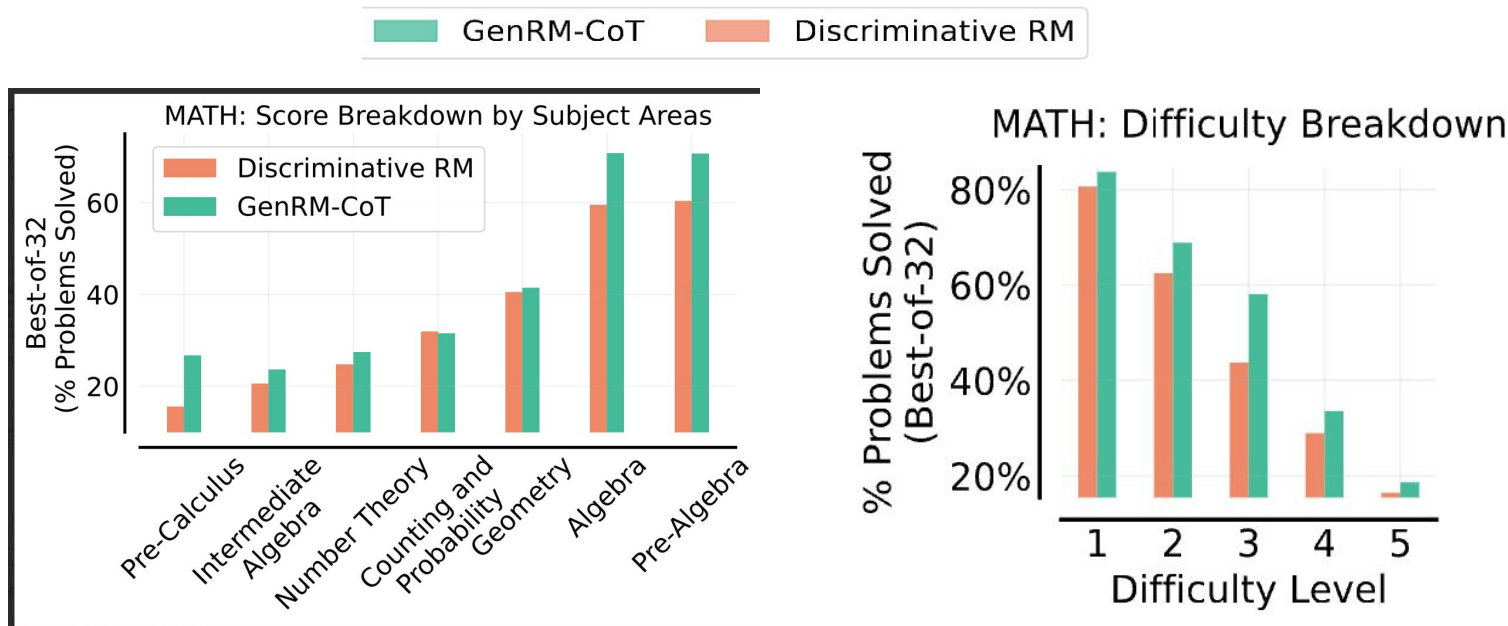
Verification: **Is the answer correct (Yes/No)? No**

GenRM-CoT (Majority Voting) score: **0.0015**

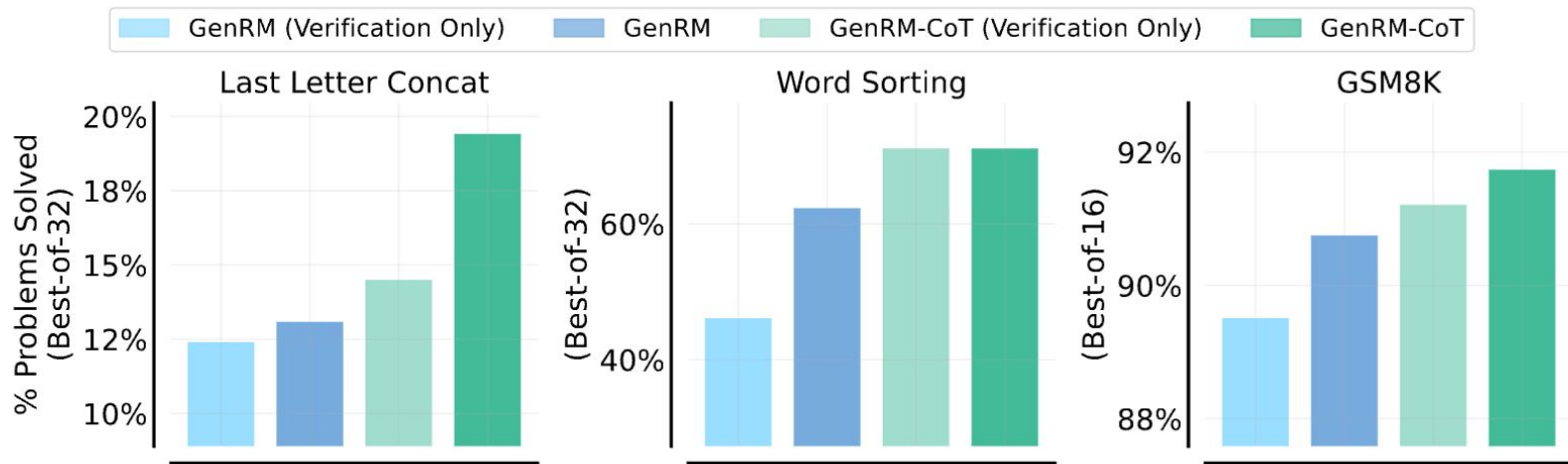
Generative Verifiers Generalize from “Easy-to-Hard” Problems



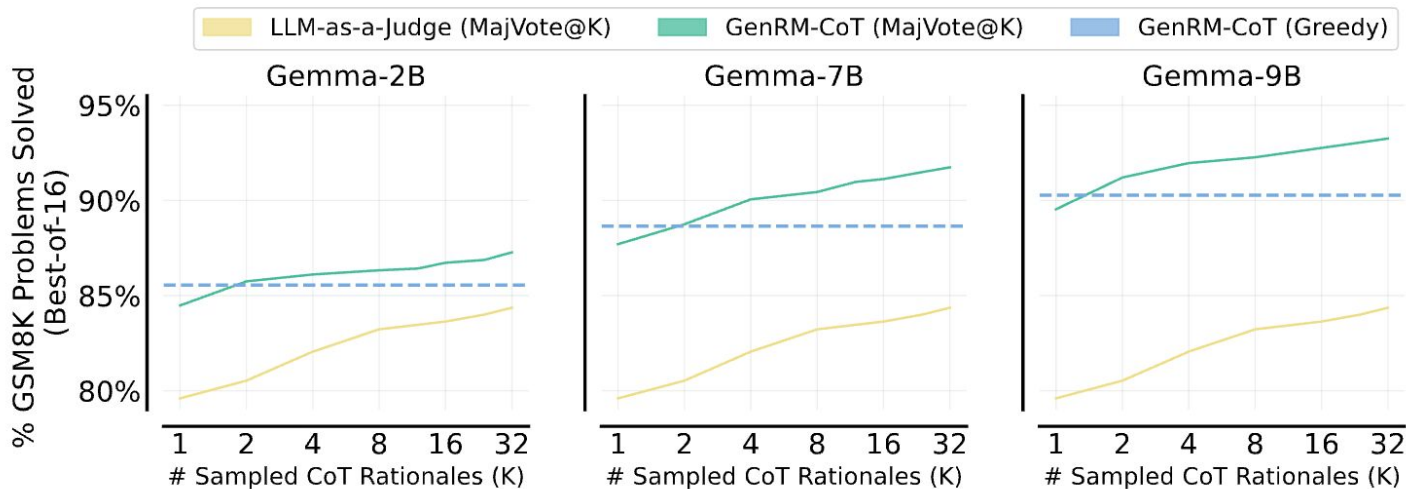
“Easy-To-Hard” Generalization on MATH



Generative Verifiers Unify Generation and Verification Tasks



Generative Verifiers Can Use Test-Time Compute (“Think More”)



Things we learned so far:

- Self-generated data improves performance, given reliable reward.
- Self-generated data can often outperform human data – it's more in-distribution!
- We can train a verifier, using both correct and incorrect solutions.
- Verifiers can help make use of test-time compute.
- Training Verifiers with Next Token Prediction has a lot of benefits!

Revisiting ReST^{EM} again!

Repeat this process a few times:

1. **Generate samples from the LLM** and filter them using binary feedback.
2. Fine-tune the model on these samples

What if we also have access to a smaller language model?

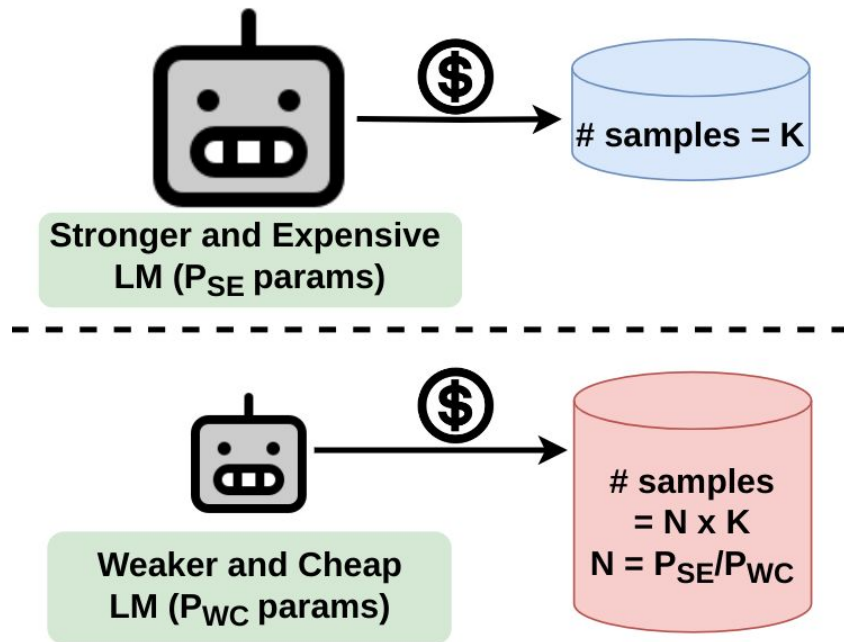
Compute-Matched Sampling

For autoregressive language models,

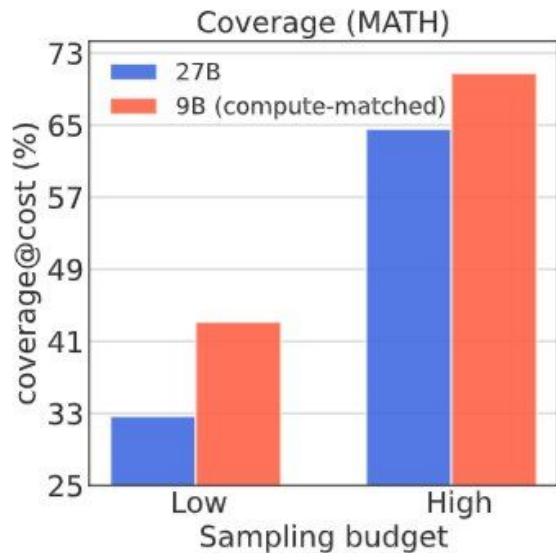
Sampling cost (FLOPs) $\sim 2ND$

N is the number of model parameters

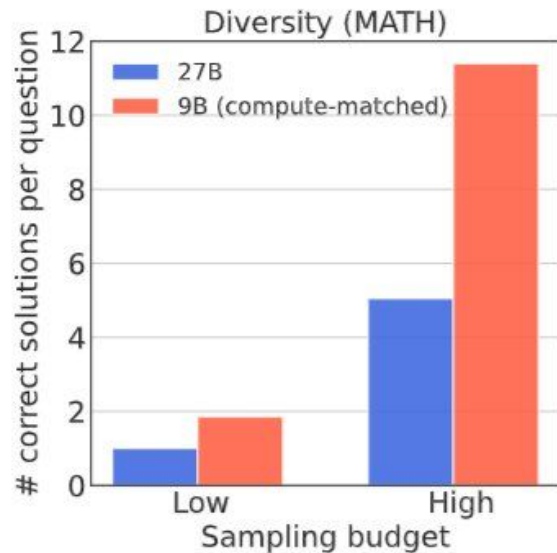
D is the number of inference tokens



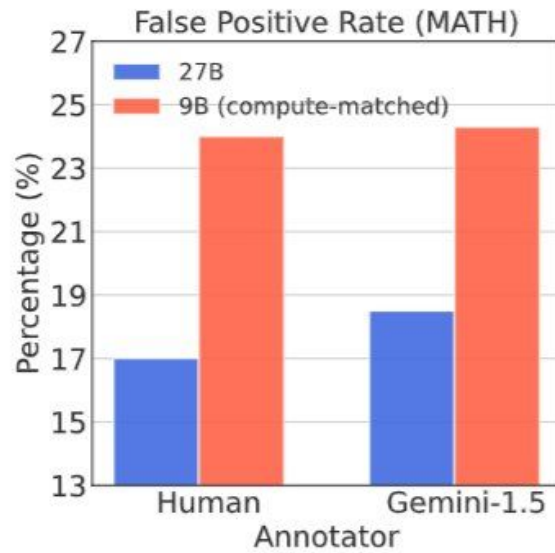
Compute-Matched Sampling Tradeoffs



(a) Coverage on MATH.

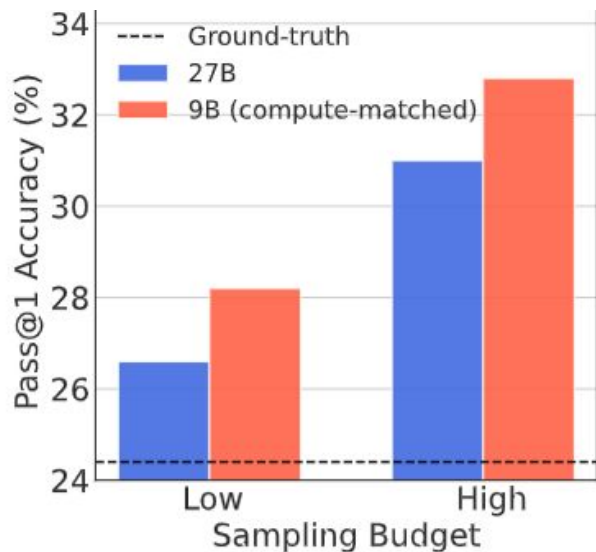


(b) Diversity on MATH.

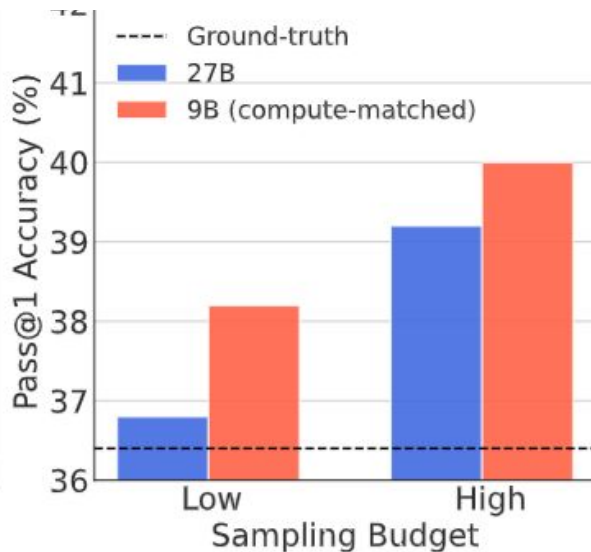


(c) False Positive Rate on MATH.

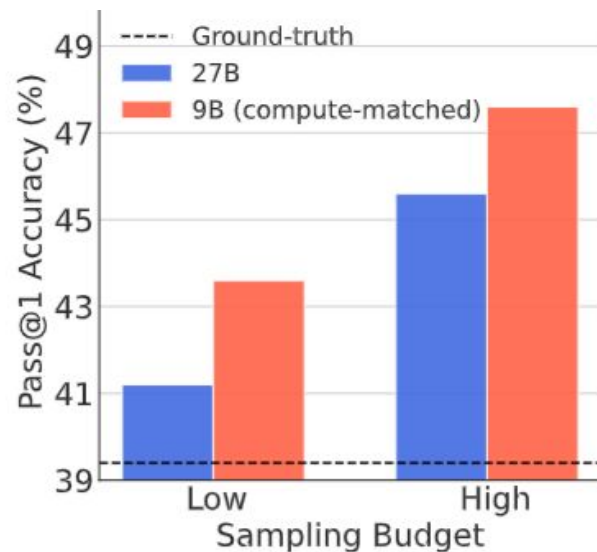
Compute-Matched Sampling Is Better!



(a) Finetuning Gemma-7B.



(b) Finetuning Gemma2-9B.

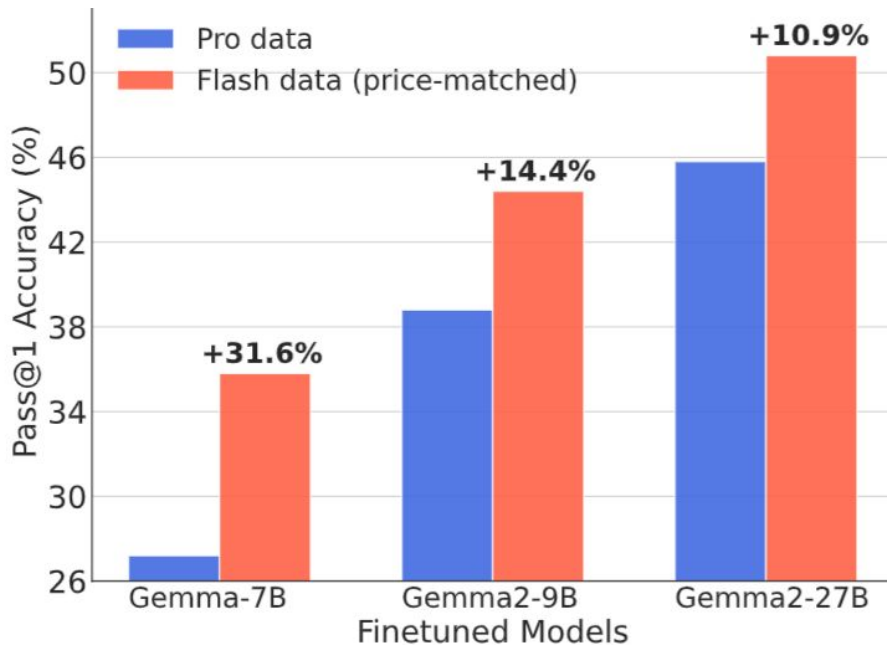


(c) Finetuning Gemma2-27B.

Cost-Matched Sampling is Even Better!

**Price of Gemini 1.5 Flash =
35x Price of Gemini 1.5 Pro**

**Knowledge distillation:
Gemma-7B, 9B, and 27B**



Things we learned so far:

- Self-generated data improves performance, given reliable reward.
- Self-generated data can often outperform human data – it's more in-distribution!
- We can train a verifier, using both correct and incorrect solutions.
- Verifiers can help make use of test-time compute.
- Training Verifiers with Next Token Prediction has a lot of benefits!
- **Consider whether a smaller model can generate better synthetic data for a given amount of compute?**

Revisiting ReST^{EM} (yet again!)

Repeat this process a few times:

1. Generate samples from the model and filter them using binary feedback.
2. **Fine-tune** the model on these samples

Is fine-tuning necessary? Wait, what?

Background: In-Context Learning

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

The diagram illustrates the structure of a prompt for in-context learning. It consists of five lines of text, each preceded by a number in a light blue vertical bar. The first line is the task description. The next three lines are examples of the task. The final line is the prompt, which is followed by a dashed line indicating the model's output. Arrows on the right side of the text point to these components: 'task description' for the first line, 'examples' for the next three lines (indicated by a bracket), and 'prompt' for the final line.

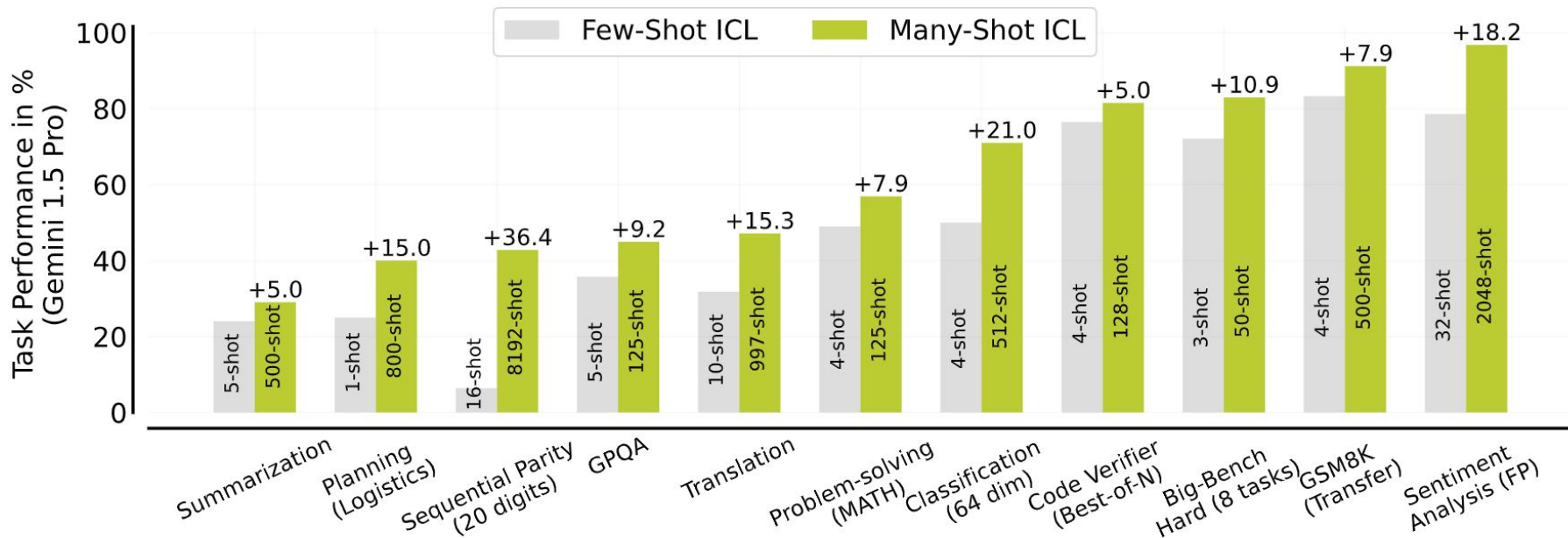
```
1  Translate English to French:
2  sea otter => loutre de mer
3  peppermint => menthe poivrée
4  plush girafe => girafe peluche
5  cheese => .....
```

← *task description*

← *examples*

← *prompt*

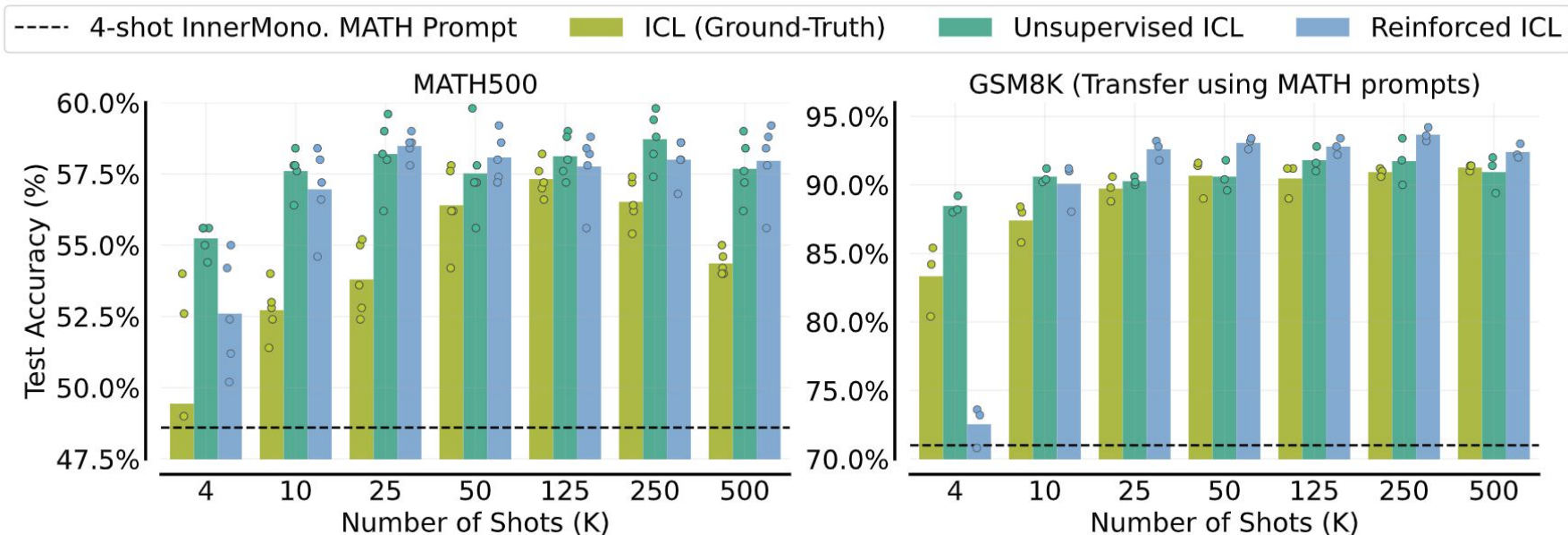
Many-Shot In-Context Learning



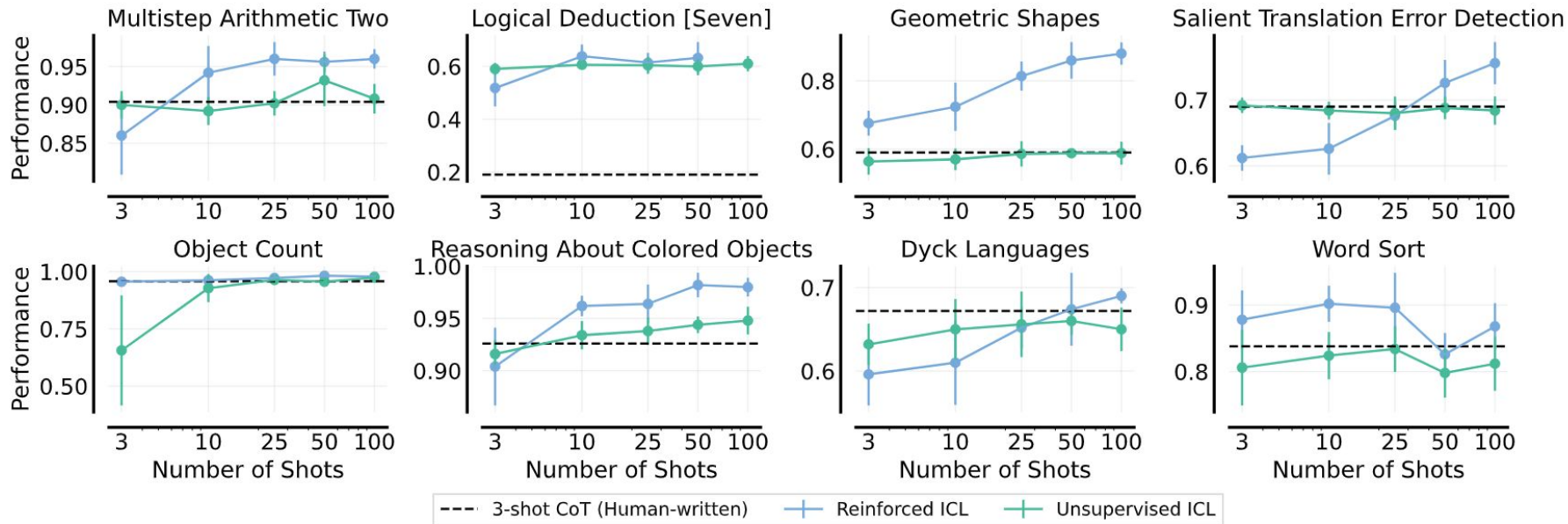
In-Context ReST^{EM}: Reinforced ICL

1. Generate samples from the model and filter them using binary feedback.
2. Put these (problem, solution) pairs in-context for the model.

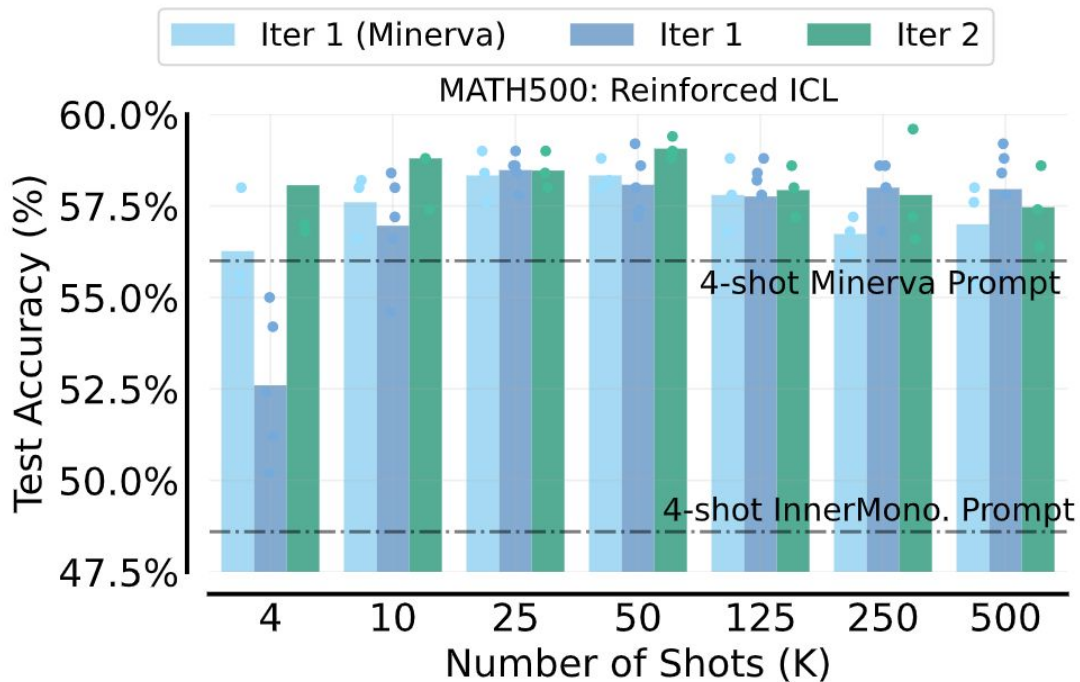
Reinforced ICL on MATH



Reinforced ICL on Big-Bench Hard



Reinforced ICL: Iteration 2



Thank you!
Questions?