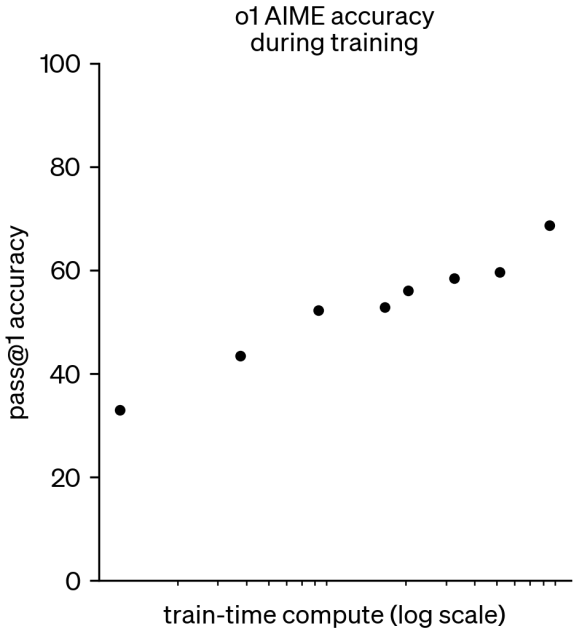
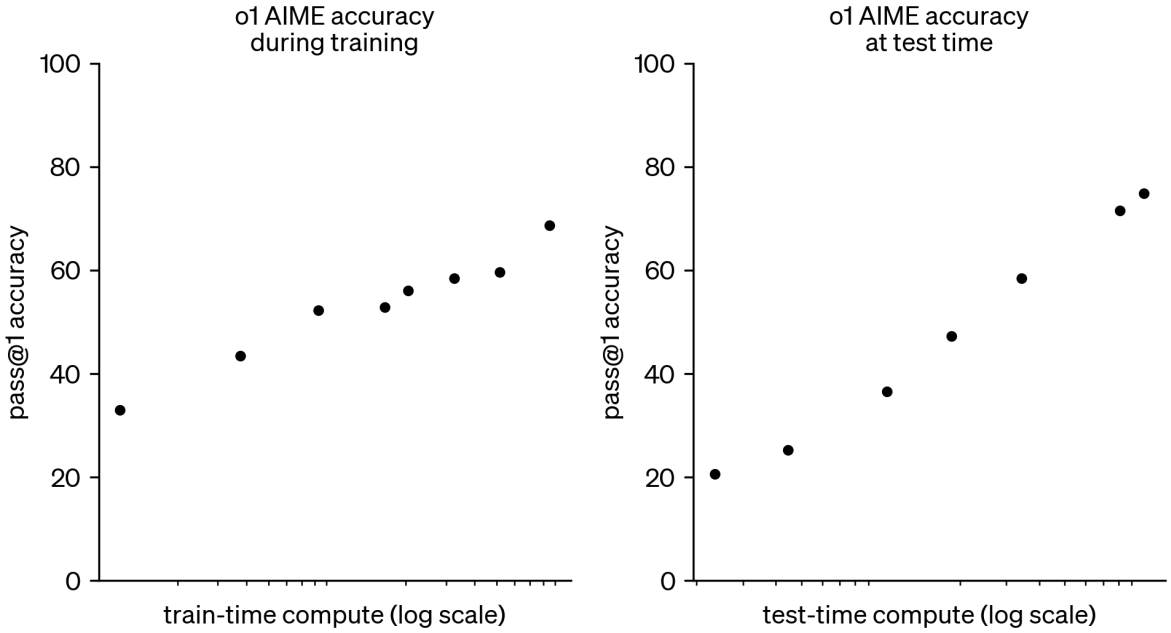
Test-time Scaling & Reasoning

Recap: scaling train-time compute



OpenAI blog post for o1

Today: scaling **test**-time compute



OpenAI blog post for o1

Today's agenda

Last lecture, we talked about **generation** (e.g. top-k sampling, top-p sampling, speculative decoding)

Now, let's:

- Generate multiple times
- Generate longer outputs

Generate multiple times

Too expensive to keep training your model?



Not enough data to keep training?



No access to model weights or output probabilities?

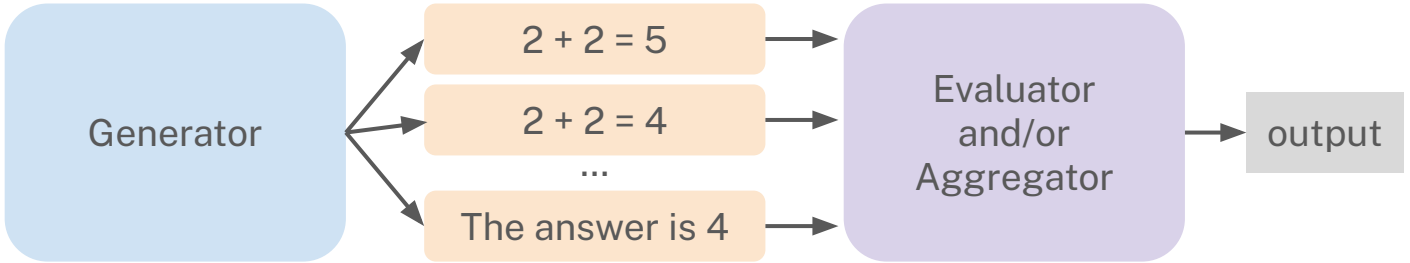


No problem!



Meta-generation algorithms

a.k.a. strategies for generating from an LM multiple times:

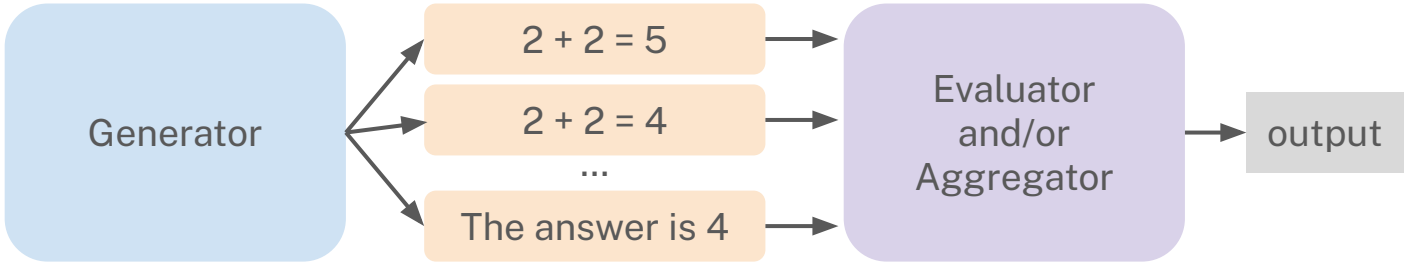


Two common patterns:

- Parallel (illustrated above), e.g. best-of-N, voting
- Refinement

Meta-generation algorithms

a.k.a. strategies for generating from an LM multiple times:



Two common patterns:

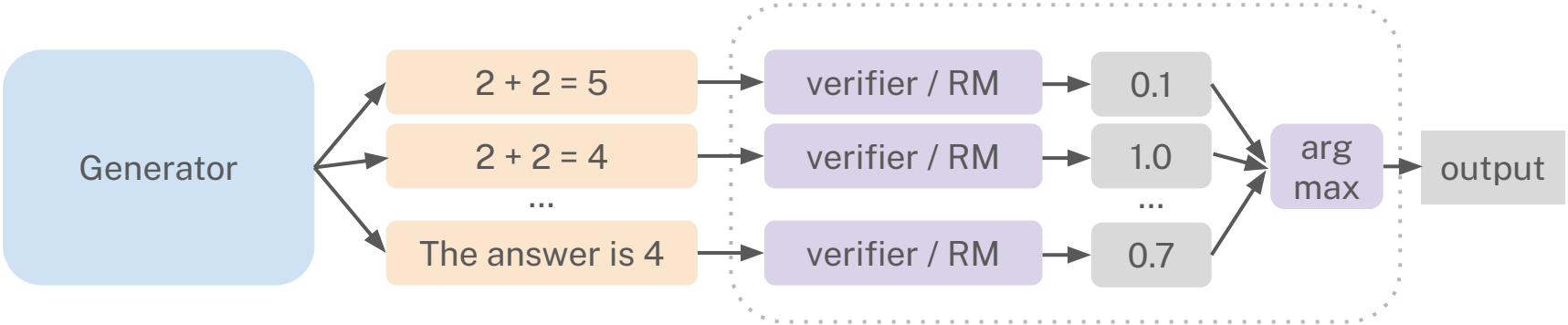
- **Parallel (illustrated above), e.g. best-of-N, voting**
- Refinement

Parallel: Best-of-N

Evaluator: a reward model or verifier *v* outputting scores

Aggregator: max

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



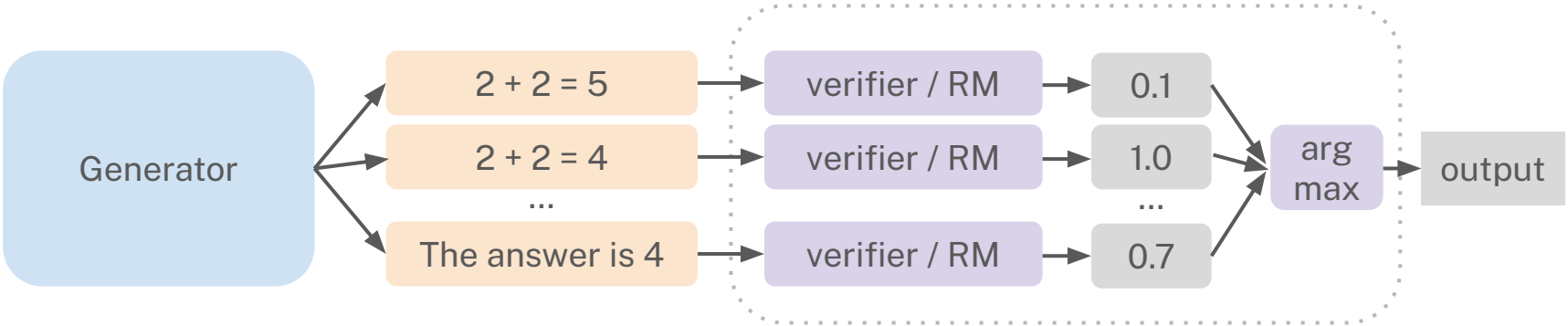
Parallel: Best-of-N

Evaluator: a reward model or verifier *v* outputting scores

Aggregator: max

$$\arg\max_{y^{(i)}} v\left(y^{(i)}\right) \approx \arg\max_{y} v\left(y\right) \approx \arg\max_{y} R\left(y\right)$$

Approximates the max true reward *R* as *N* increases, if *v* is a good reward model!



Types of verifiers

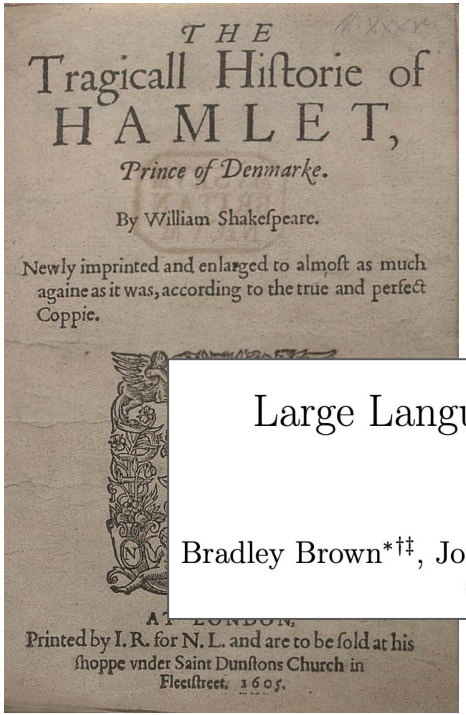
They might evaluate:

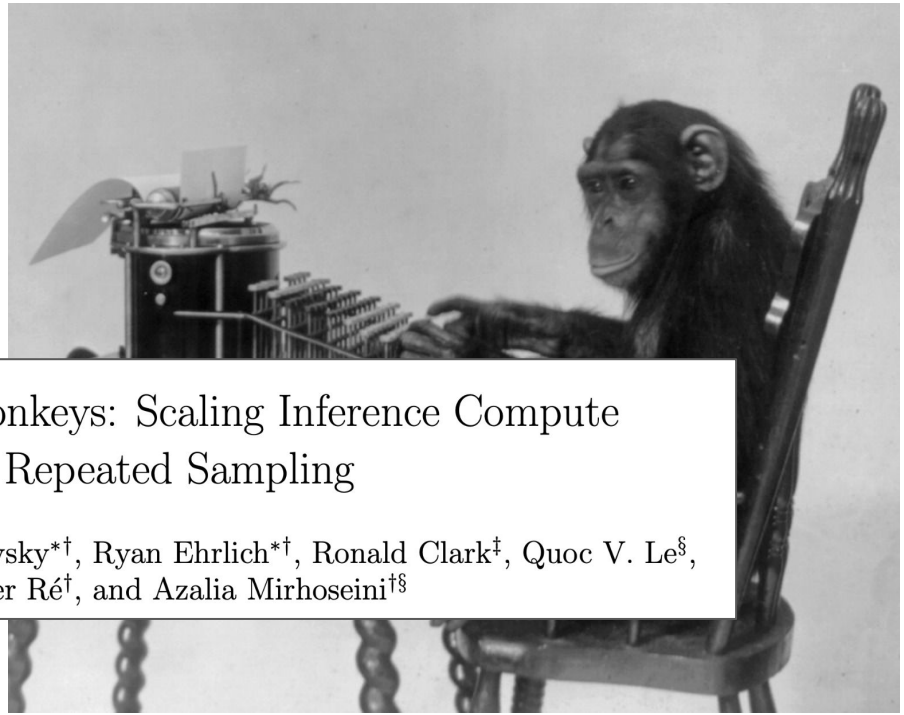
- Outcome, e.g. is the final answer correct?
- Process, e.g. is the partial solution correct, or might it lead to the correct solution?

Implementations:

- Just prompt an LM (maybe even the same one as you use to generate)
- Train one
- Environment-specific feedback (e.g. unit tests for code)

Infinite monkey theorem


THE
Tragicall Hiſtorie of
HAMLET,
Prince of Denmarke.
By William Shakeſpeare.
Newly imprinted and enlarged to almoſt as much againe as it was, according to the true and perfect Coppie.
Printed by I. R. for N. L. and are to be ſold at his ſhoppe vnder Saint Dunſtons Church in Fleetſtreet. 1605.

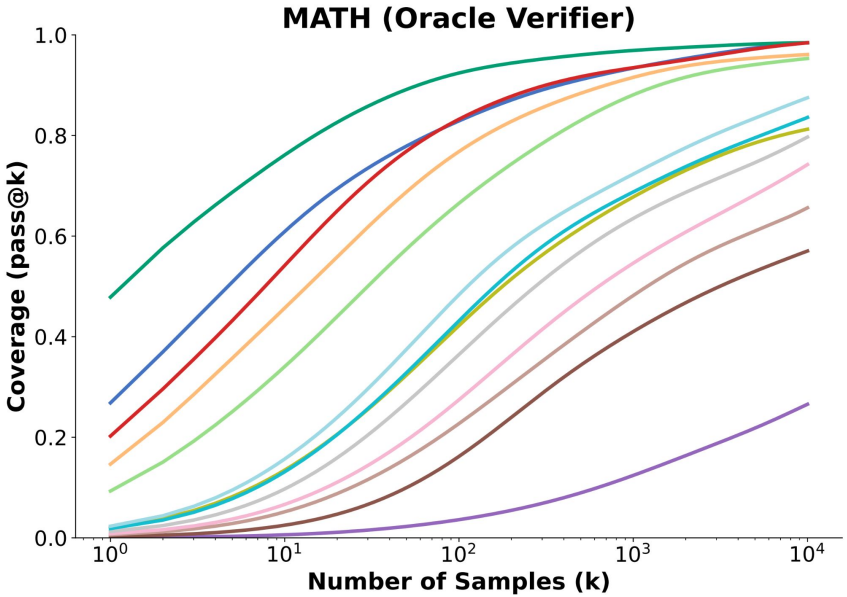


Large Language Monkeys: Scaling Inference Compute with Repeated Sampling

Bradley Brown*†‡, Jordan Juravsky*†, Ryan Ehrlich*†, Ronald Clark‡, Quoc V. Le§, Christopher Ré†, and Azalia Mirhoseini†§

Images from Wikipedia

What if our verifier was oracle? 🧚



Pass@k

The % of problems solved when we take *k* samples from the model for each problem, and submit all of them for eval.

If any solution succeeds, we pass!

Large Language Monkeys paper

Training a verifier

Example: GSM8k paper (Cobbe et al. 2021)

Train a verifier (reward model) to classify whether a solution *S* to a grade school math word problem *Q* is correct.

```
Natalia sold clips to 48 of her
friends in April, and then she
sold half as many clips in May.
How many clips did Natalia sell
altogether in April and May?

Natalia sold 48/2 =
<<48/2=24>>24 clips in May.

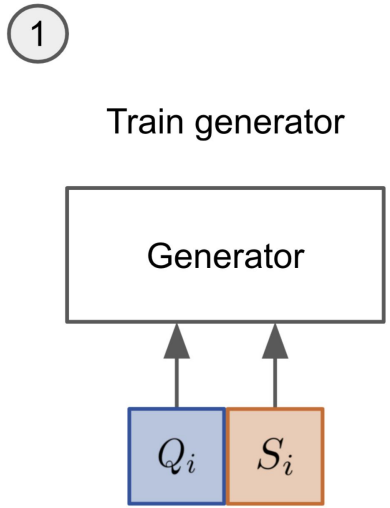
Natalia sold 48+24 =
<<48+24=72>>72 clips altogether
in April and May.

Final Answer: 72.
```

Training a verifier

Example: GSM8k paper (Cobbe et al. 2021)

Train a verifier (reward model) to classify whether a solution *S* to a grade school math word problem *Q* is correct.

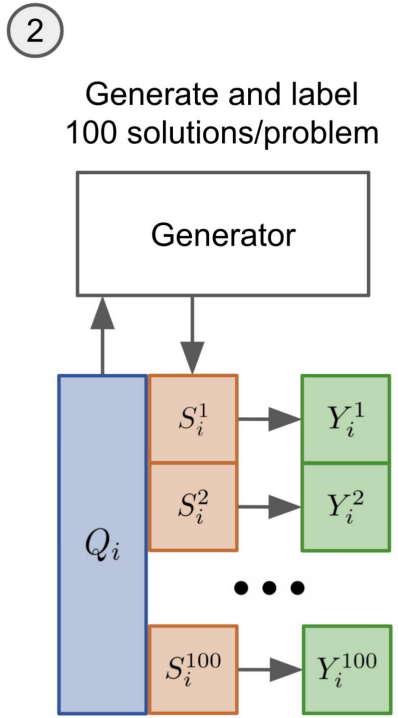


Finetune a model (the “generator”) for 2 epochs on the training set.

Training a verifier

Example: GSM8k paper (Cobbe et al. 2021)

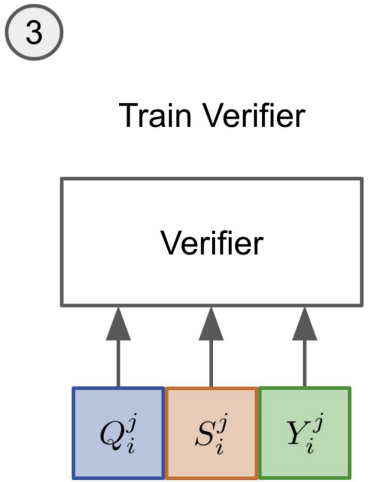
Train a verifier (reward model) to classify whether a solution *S* to a grade school math word problem *Q* is correct.



Training a verifier

Example: GSM8k paper (Cobbe et al. 2021)

Train a verifier (reward model) to classify whether a solution *S* to a grade school math word problem *Q* is correct.



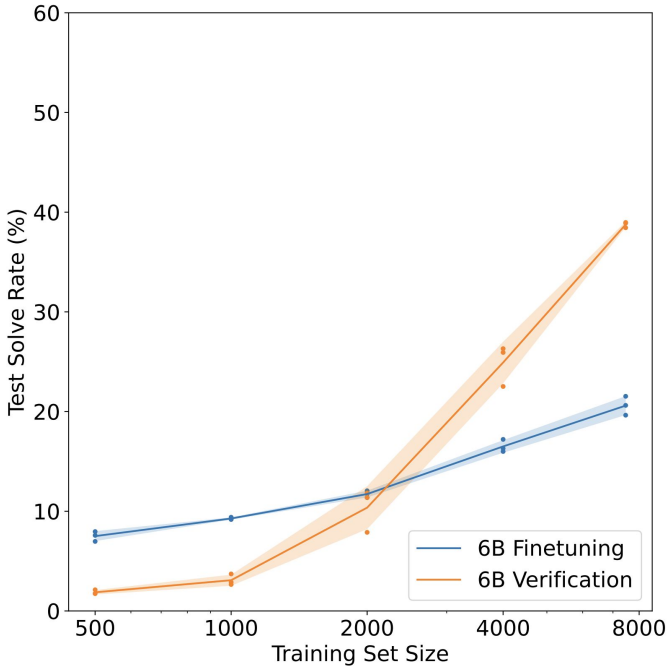
Train a verifier for a single epoch on this dataset of (questions, solution, correctness).

Training a verifier

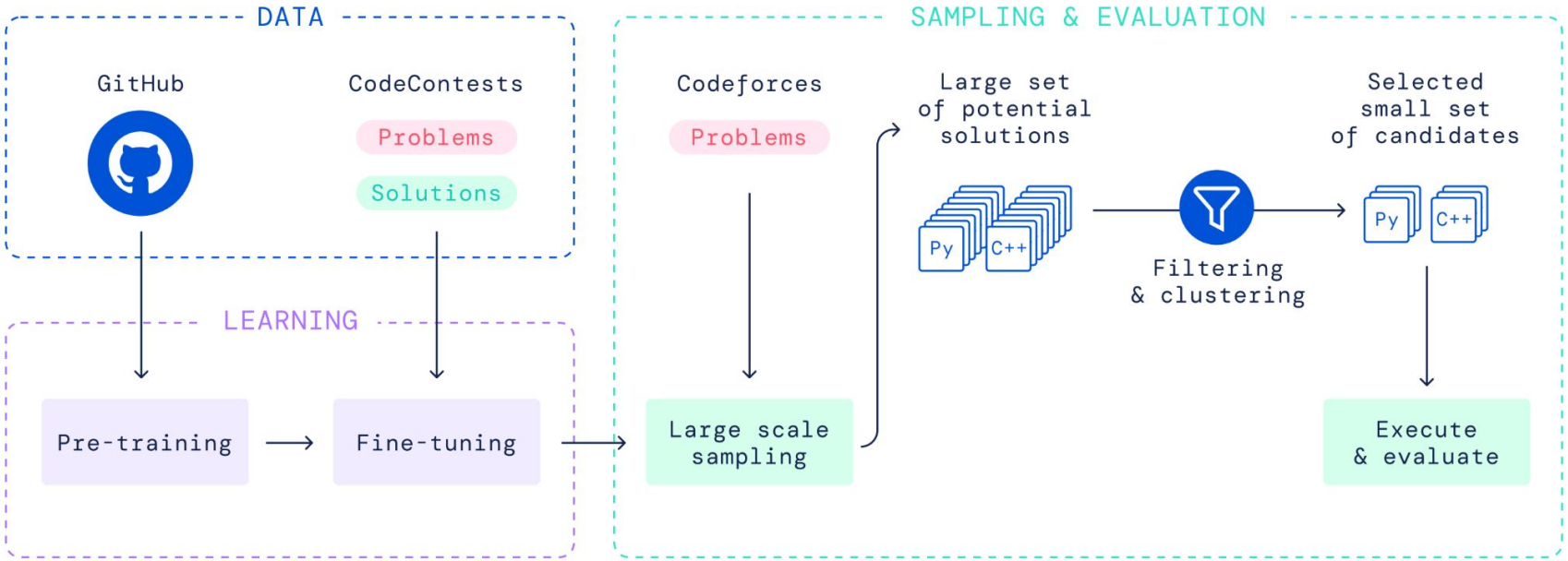
Example: GSM8k paper (Cobbe et al. 2021)

Train a verifier (reward model) to classify whether a solution *S* to a grade school math word problem *Q* is correct.

At test time: sample 100 solutions from a generating LM, and return the one with the highest verifier score.

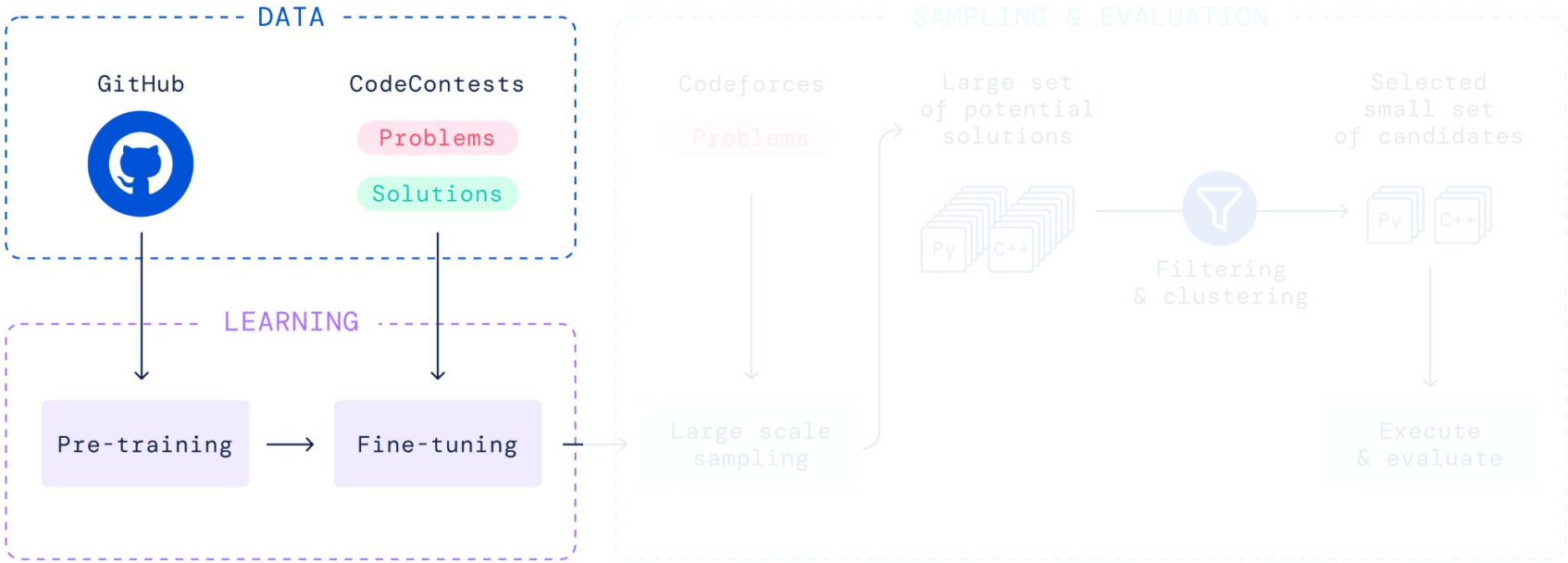


Another Best-of-N example: Alphacode



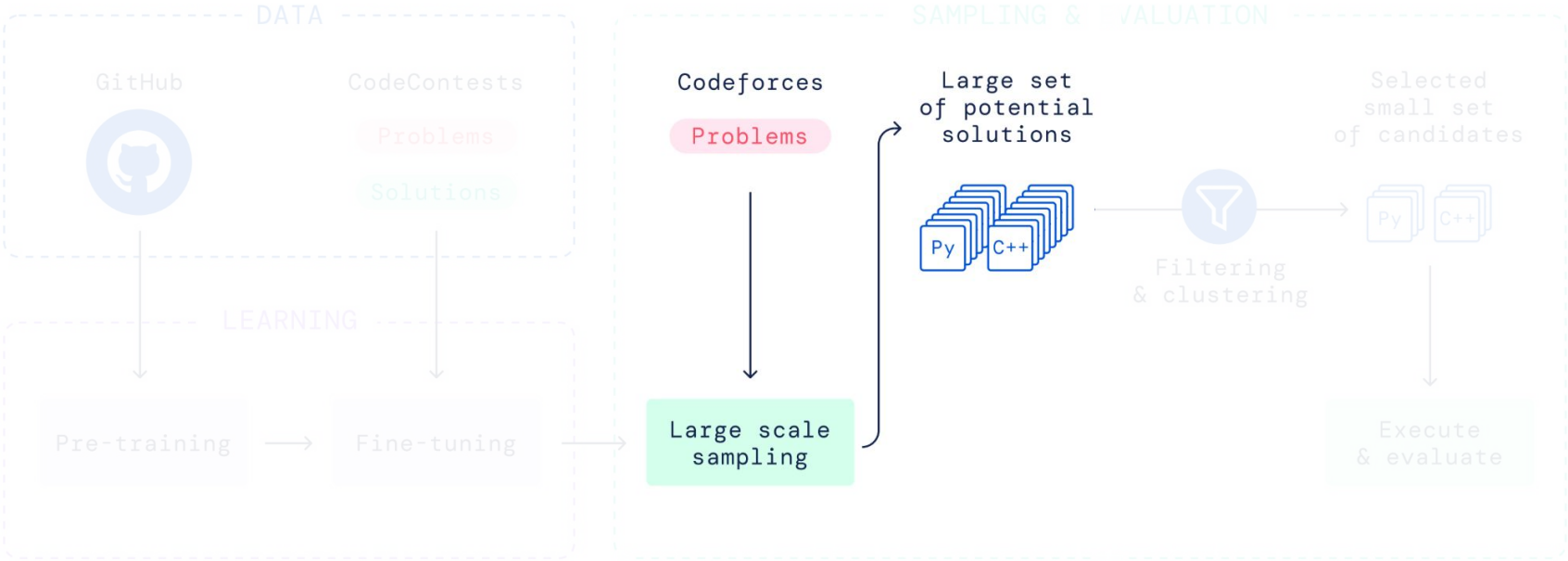
Li et al. 2022

Another Best-of-N example: Alphacode



Li et al. 2022

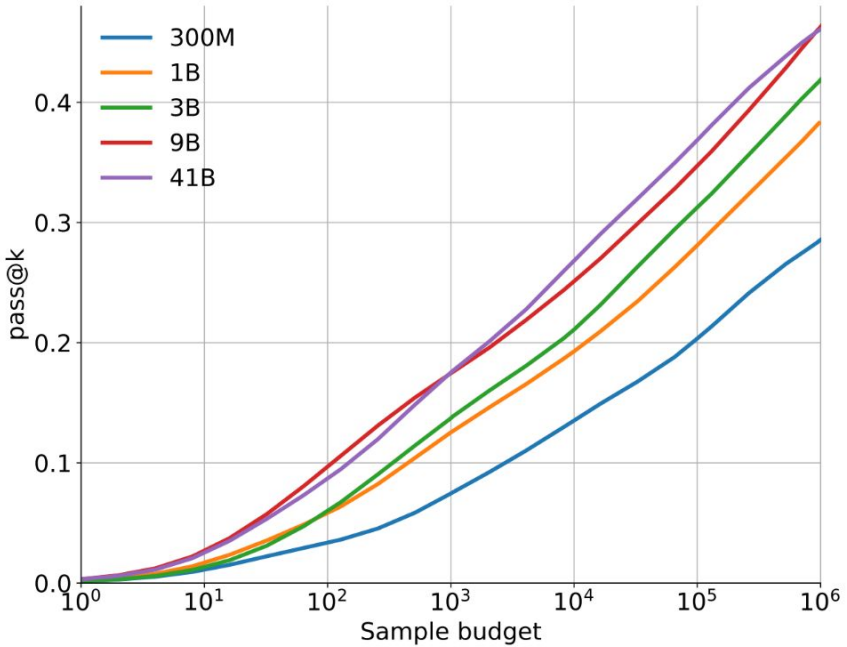
Another Best-of-N example: Alphacode



Li et al. 2022

Alphacode, w/ unlimited attempts

Whether any solution in our sample solves the problem

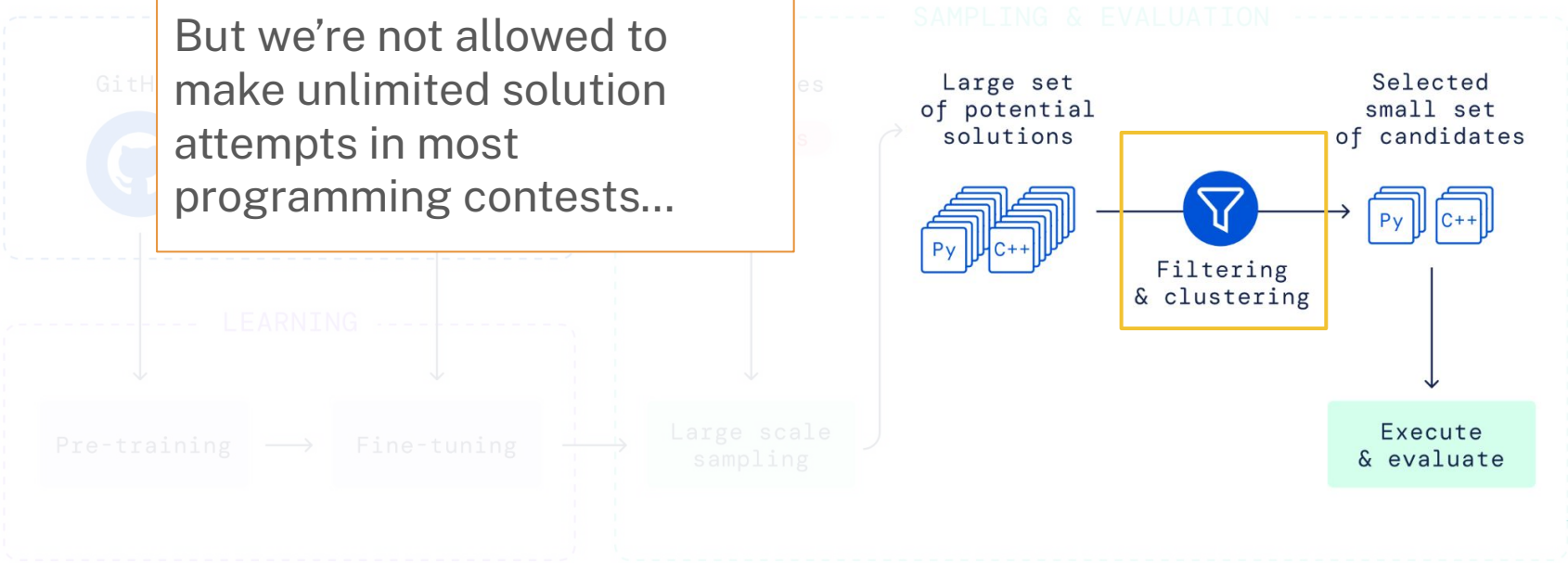


Larger models require fewer samples to achieve the same solve rate.

Log-linear

Another Best-of-N example: Alphacode

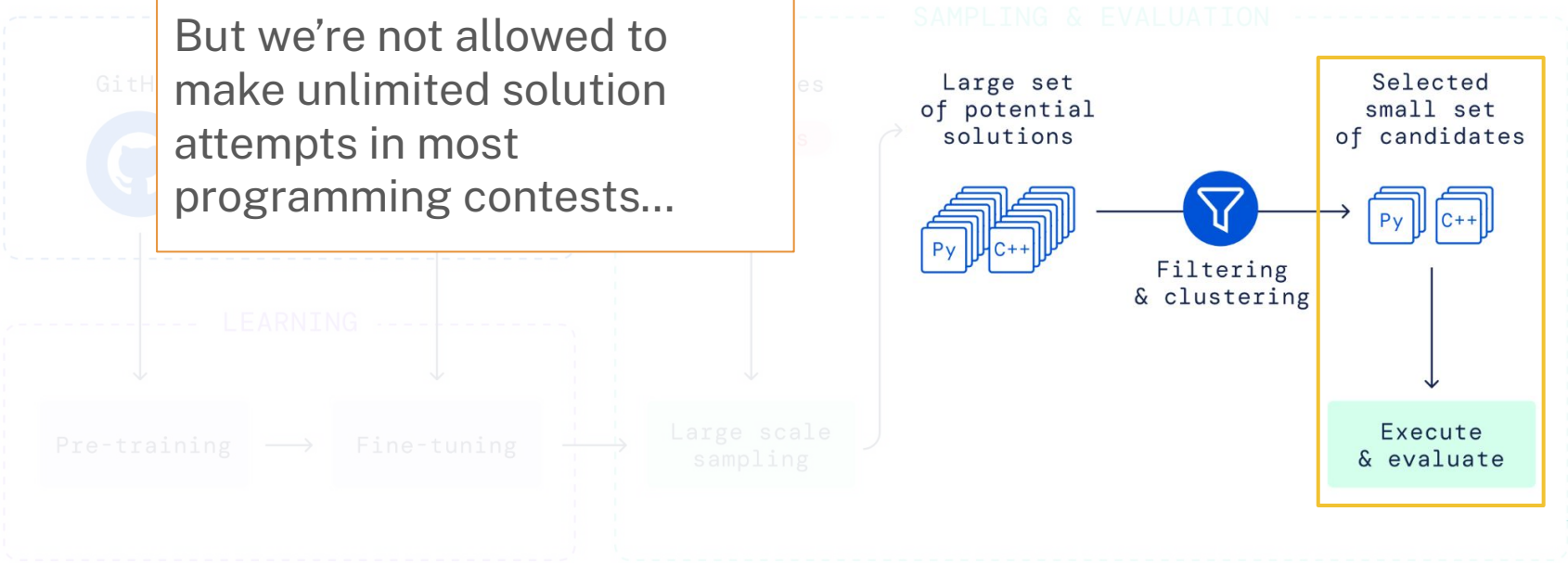
But we're not allowed to make unlimited solution attempts in most programming contests...



Li et al. 2022

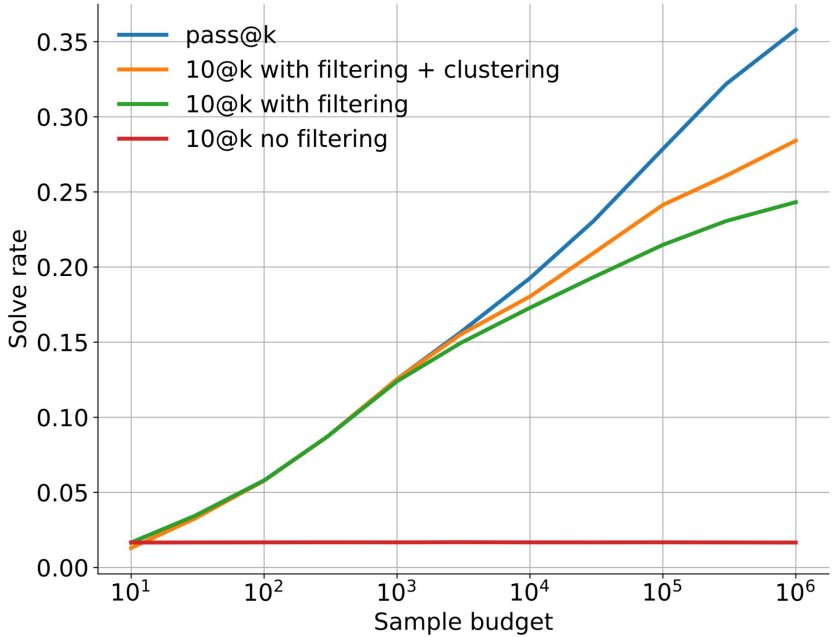
Another Best-of-N example: Alphacode

But we're not allowed to make unlimited solution attempts in most programming contests...



Li et al. 2022

Another Best-of-N example: Alphacode

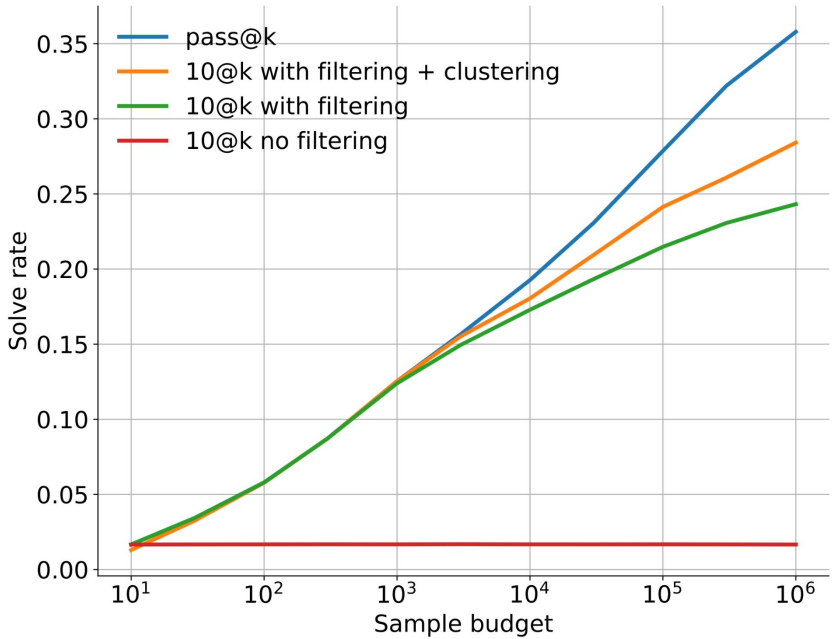


Li et al. 2022

10@k

The % of problems solved when we take *k* samples from the model for each problem, but we can only submit 10 of them for eval.

Another Best-of-N example: Alphacode



Can submit all solutions, no sampling

Sample 10 from clusters of filtered set

Sample 10 from a filtered set

Randomly sample 10 with no filtering

Li et al. 2022

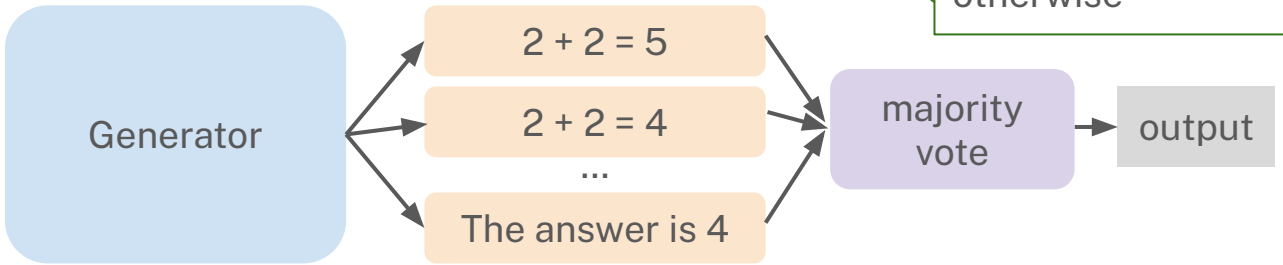
Parallel: Voting, a.k.a. self-consistency

Aggregation: pick the most common answer

No reward model! Though you'll need some notion of output equivalence

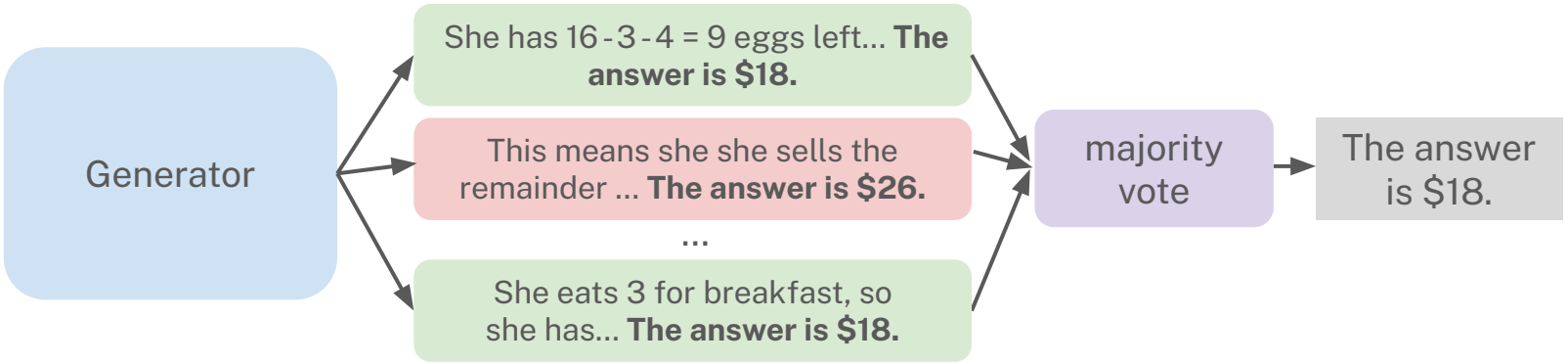
$$\arg\max_a \sum_{i=1}^{N} 1\left[y^{(i)} = a\right]$$

Indicator function where we get 1 if our sampled output equals *a* and 0 otherwise

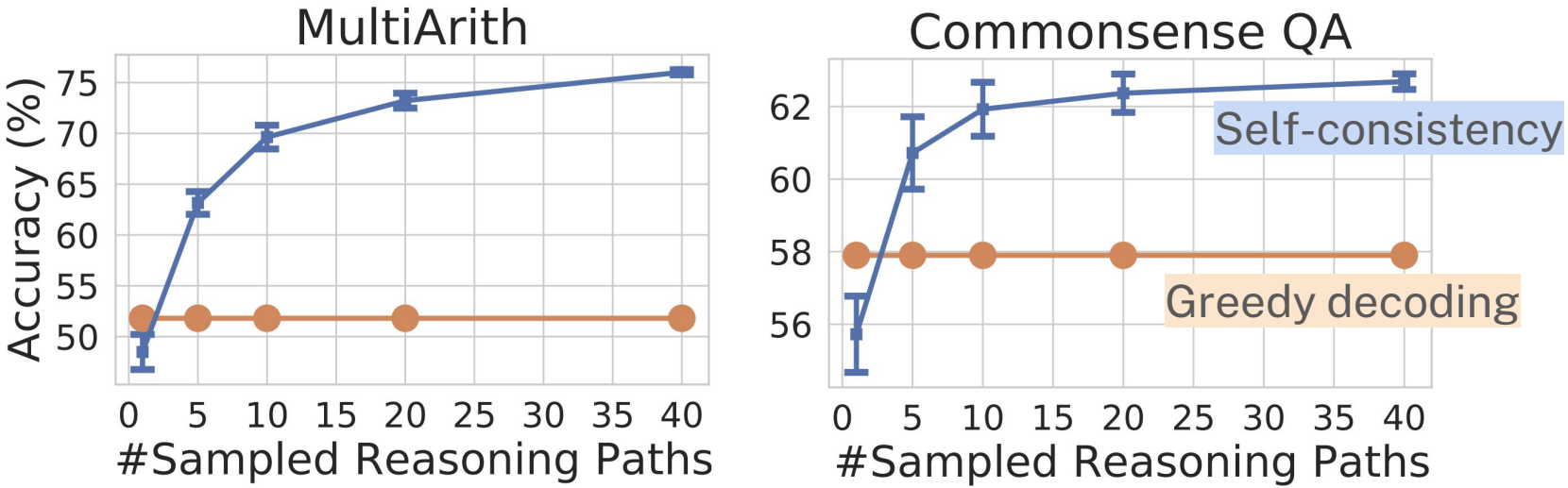


Self-consistency + CoT

“Marginalize out” reasoning chains and only compare final outputs.



Self-consistency + CoT



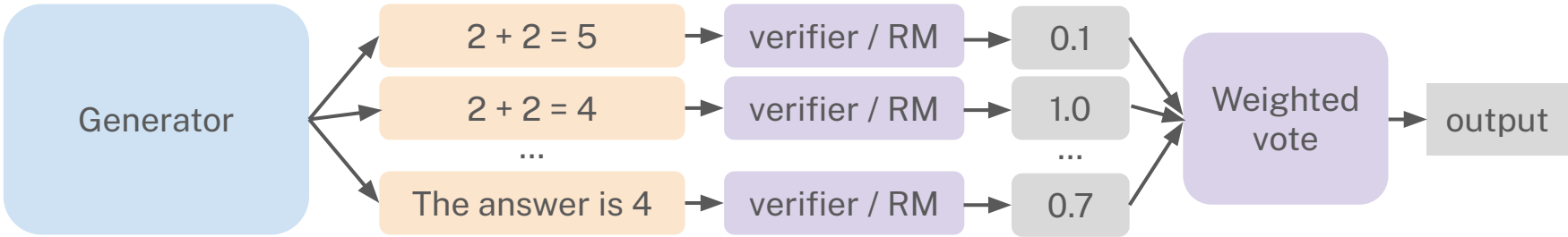
Wang et al. 2022

Parallel: Weighted Voting

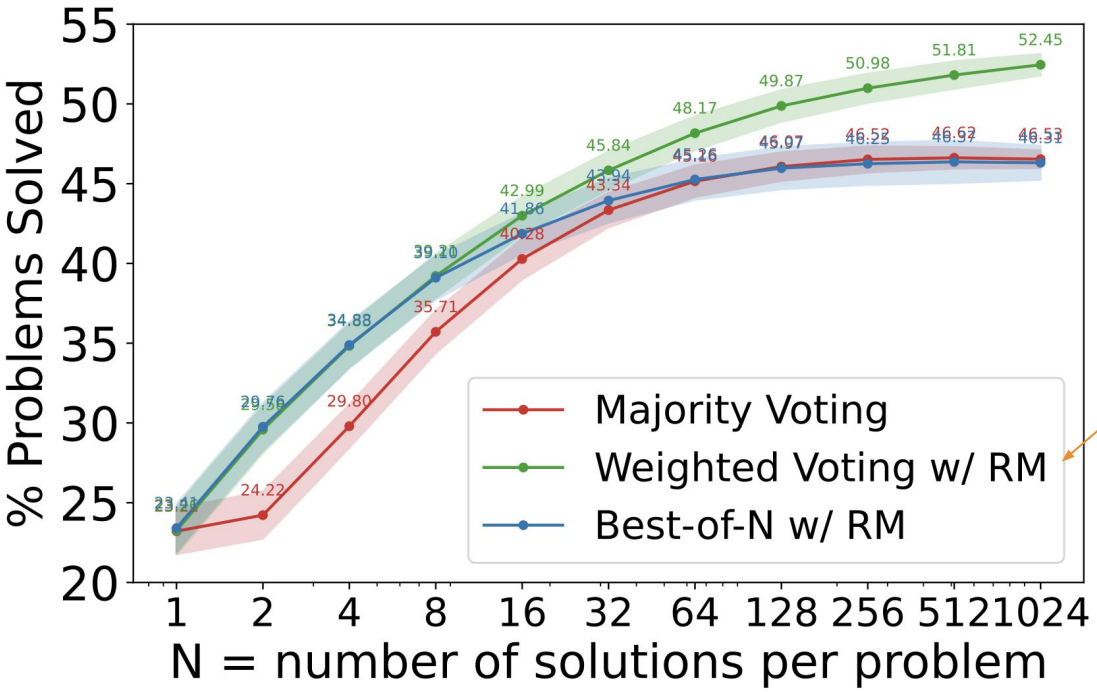
Aggregation: pick the most common answer

Scoring: use a reward model *v* to weight votes

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



Parallel: Weighted Voting



Outcome & process reward model

Sun et al. 2024

What is voting doing?

As the number of sampled candidates $\to \infty$, voting accuracy converges to:

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

Number of test examples

Verifier score or reward

Probability that an LM generates path *z* yielding answer *a* given input *x*. With more samples, we can estimate this better.

Wu et al. 2024

What is voting doing?

As the number of sampled candidates $\to \infty$, voting accuracy converges to:

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

Add one each time the bracketed condition is true.

The correct answer

The weighted majority vote answer

Wu et al. 2024

What is voting doing?

As the number of sampled candidates $\to \infty$, voting accuracy converges to:

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

The fraction of *M* test examples that we get right

Takeaway 1: With infinite samples, weighted voting gets a question right only if the correct answer has the highest reward-weighted total vote. So, we converge at some accuracy "ceiling".

Wu et al. 2024

What is voting doing?

As the number of sampled candidates $\to \infty$, voting accuracy converges to:

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

The fraction of *M* test examples that we get right

Takeaway 2: Weighted voting is better than voting only when $v \cdot g$ assigns more total mass to correct answers than g alone.

Wu et al. 2024

What is voting doing?

As the number of sampled candidates $\to \infty$, voting accuracy converges to:

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

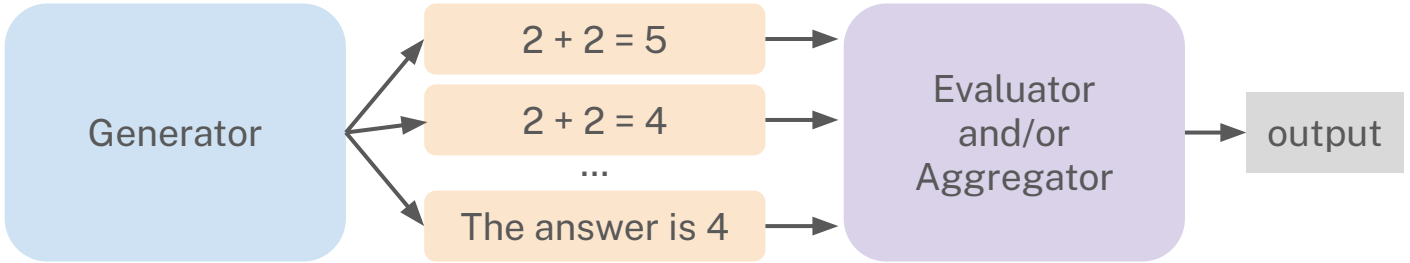
The fraction of *M* test examples that we get right

Takeaway 3: To improve performance, one has to either improve the reward model *v* or improve the generator *g*.

Wu et al. 2024

Meta-generation algorithms

a.k.a. strategies for generating from an LM multiple times:

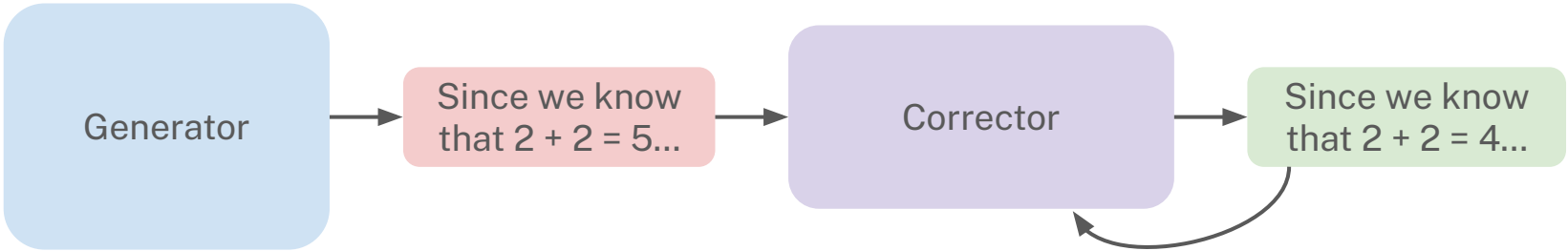


Two common patterns:

- **Parallel (illustrated above), e.g. best-of-N, voting**
- Refinement

Meta-generation algorithms

a.k.a. strategies for generating from an LM multiple times:

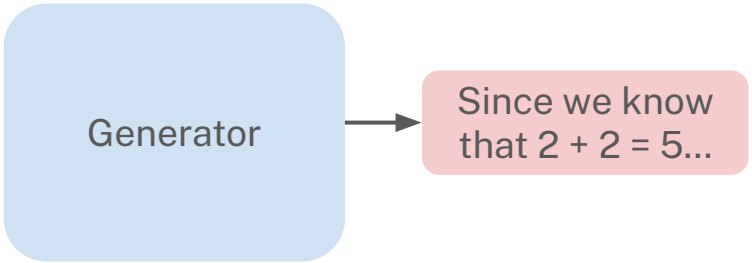


Two common patterns:

- Parallel, e.g. best-of-N, voting
- **Refinement (illustrated above)**

Meta-generation algorithms

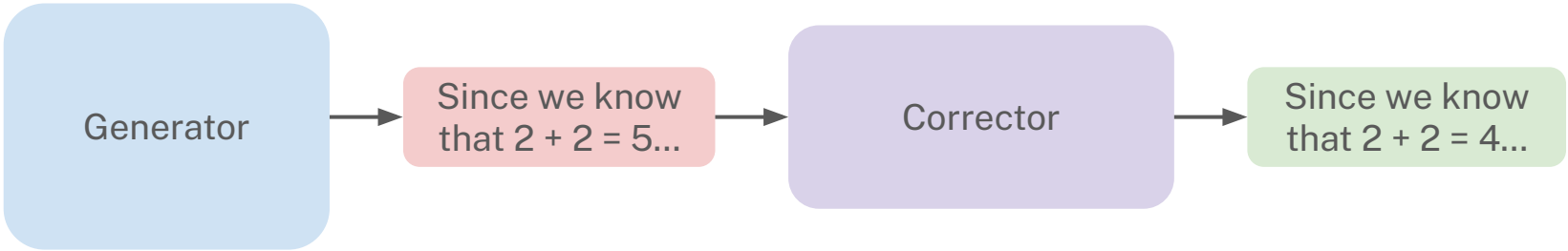
a.k.a. strategies for generating from an LM multiple times:



First, we **generate** a solution and answer to a prompt.

Meta-generation algorithms

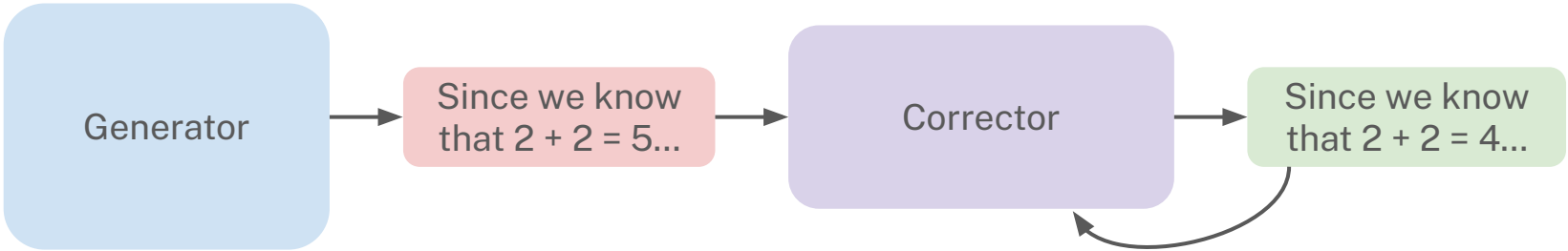
a.k.a. strategies for generating from an LM multiple times:



Then, we **verify**: look at our prompt and proposed solution/answer, and give some feedback to produce a new output

Meta-generation algorithms

a.k.a. strategies for generating from an LM multiple times:

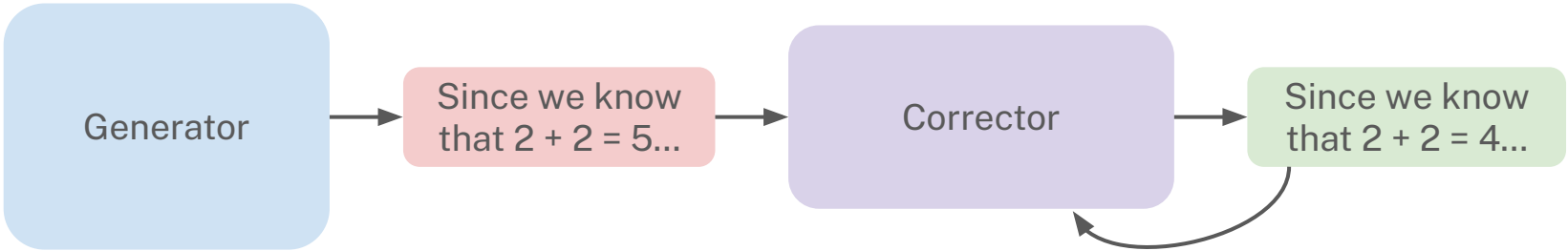


And loop!

We terminate when the verifier is “satisfied” or until some max # of iterations.

Meta-generation algorithms

a.k.a. strategies for generating from an LM multiple times:



Iterative refinement tends to work well with **accurate and noise-free** extrinsic feedback from the corrector.

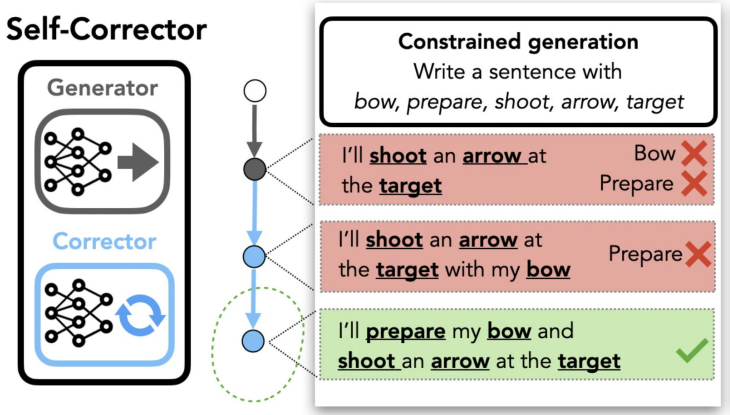
Good feedback **adds new information** and **localizes errors** so that the model has something specific to refine.

Self-refinement

Re-prompt the same model to generate, give feedback, and refine

Mixed results:

- Can work on tasks that are easy to validate or verify



Welleck et al. 2023

Self-refinement

Re-prompt the same model to generate, give feedback, and refine

Mixed results:

- Can work on tasks that are easy to validate or verify
- Less clear for harder or more complex tasks

		# calls	GSM8K	CommonSenseQA
GPT-4-Turbo	Standard Prompting	1	**91.5**	**84.0**
	Self-Correct (round 1)	3	88.0	81.5
	Self-Correct (round 2)	5	90.0	83.0
Llama-2	Standard Prompting	1	**62.0**	**64.0**
	Self-Correct (round 1)	3	43.5	37.5
	Self-Correct (round 2)	5	36.5	36.5

Huang et al. 2024

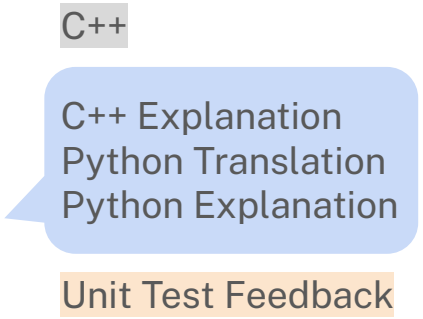
Example: code generation

Two forms of feedback:

- Unit test results (extrinsic)
- Explanation (intrinsic)

Example prompt for code translation task: "*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.*"

Loop for ≤ 10 refinement turns



Chen et al. 2024

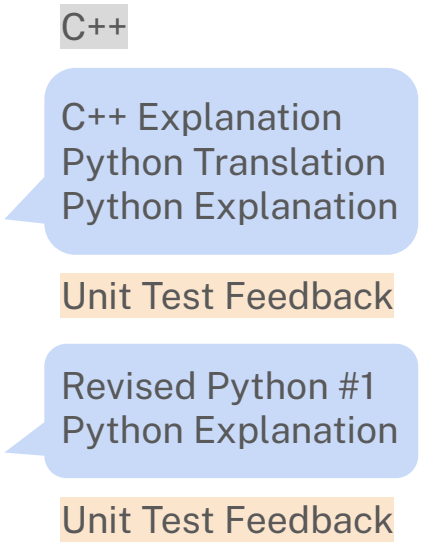
Example: code generation

Two forms of feedback:

- Unit test results (extrinsic)
- Explanation (intrinsic)

Example prompt for code translation task: "*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.*"

Loop for ≤ 10 refinement turns



Chen et al. 2024

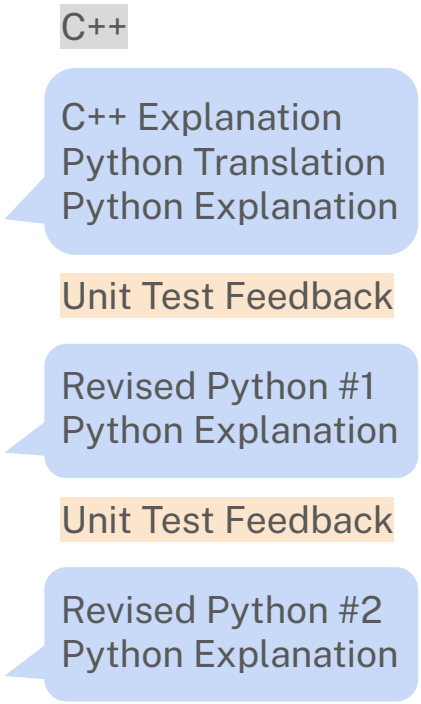
Example: code generation

Two forms of feedback:

- Unit test results (extrinsic)
- Explanation (intrinsic)

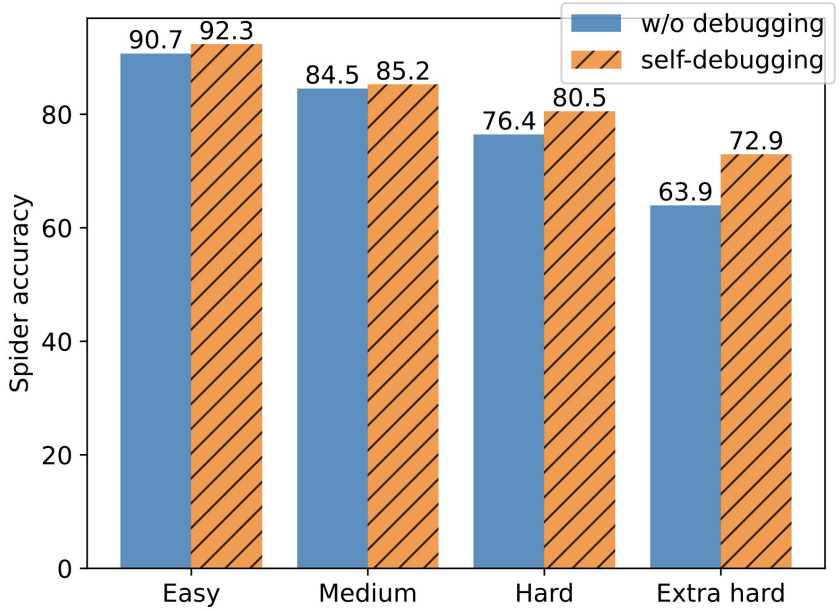
Example prompt for code translation task: “*Below are C++ programs with incorrect Python translations. Explain the original code, then explain the translations line by line and correct them using the provided feedback.*”

Loop for ≤ 10 refinement turns



Chen et al. 2024

Example: code generation



Chen et al. 2024

Inference scaling laws

Compute is a function of:

model size

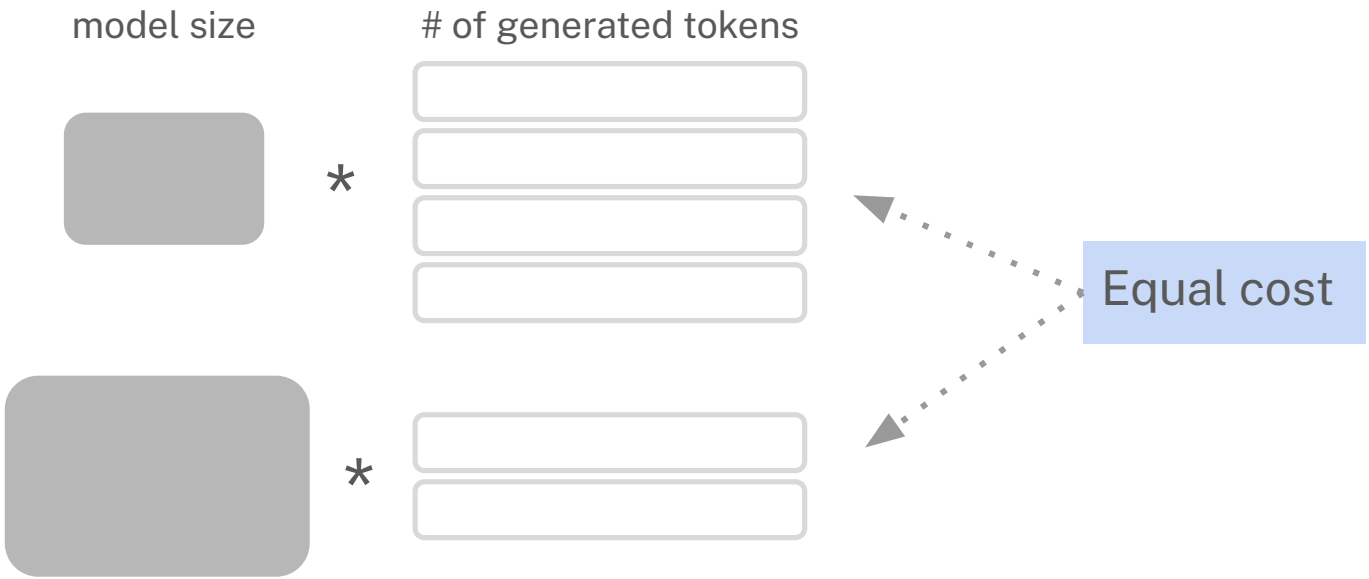


*

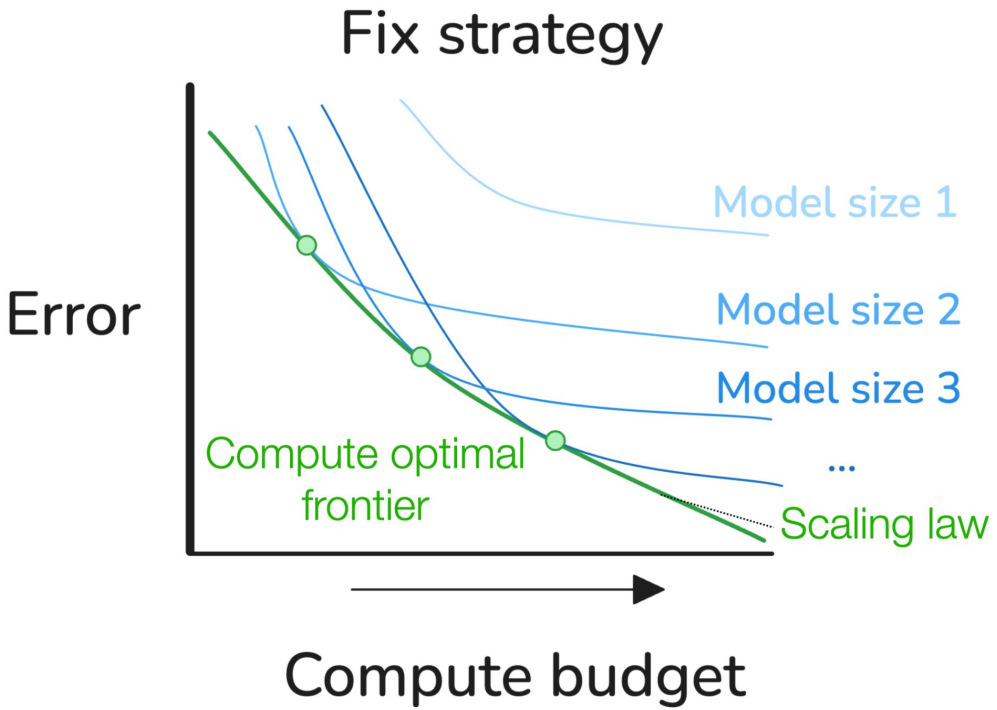
# of generated tokens

Inference scaling laws

We can increase either or both of these:

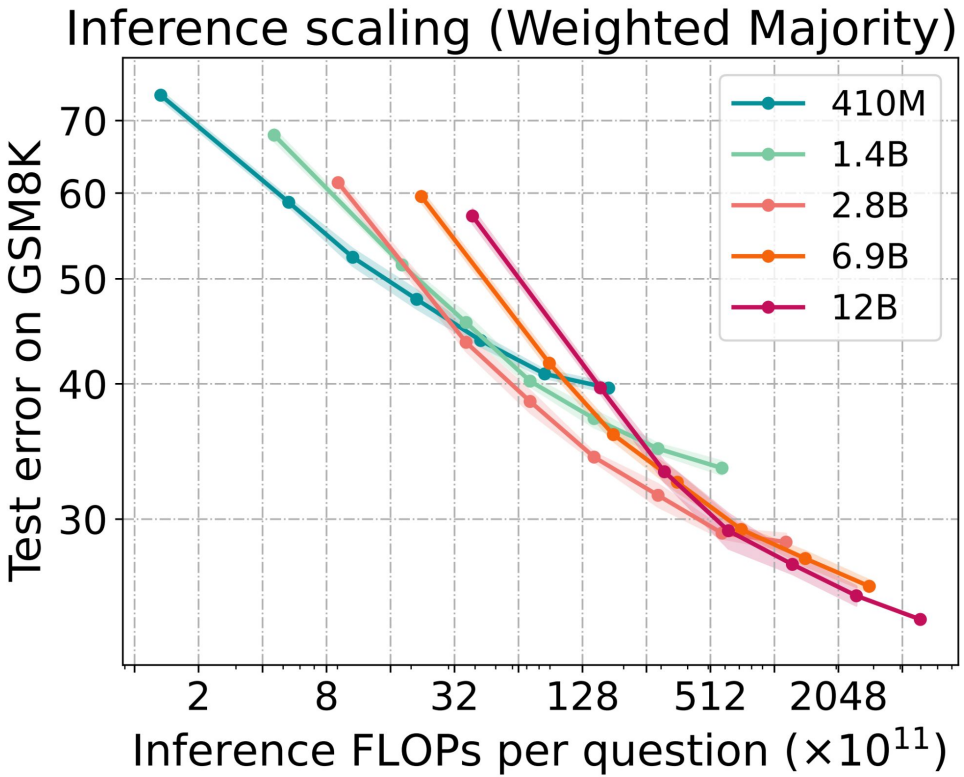


Inference scaling laws



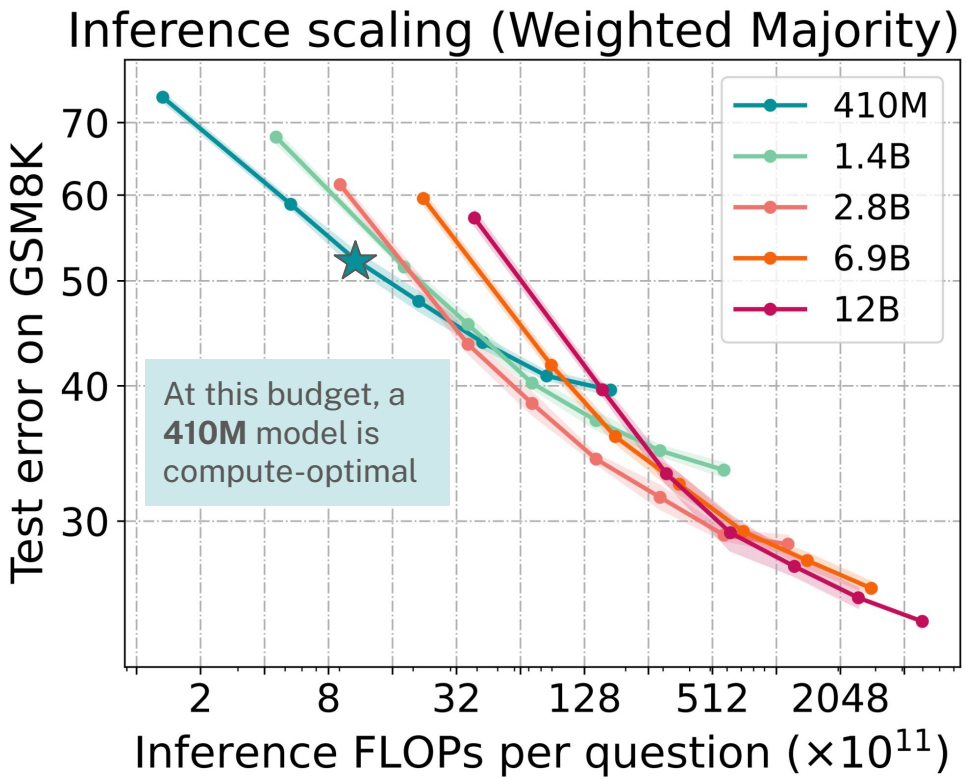
From Sean Welleck's slides

Inference scaling laws



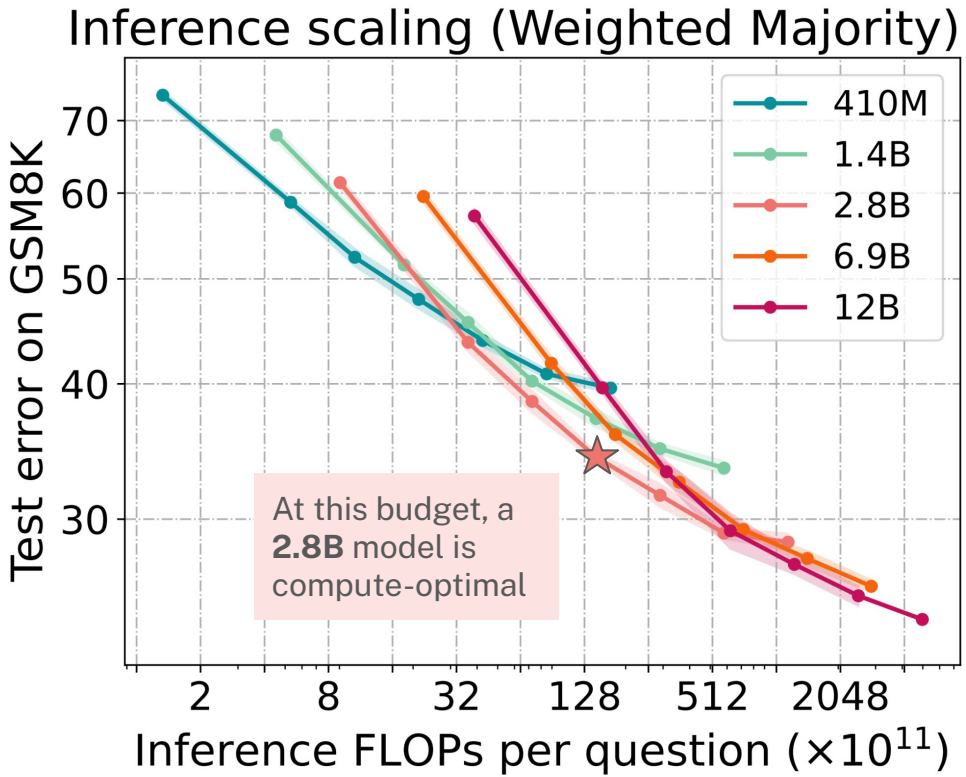
Wu et al. 2024

Inference scaling laws



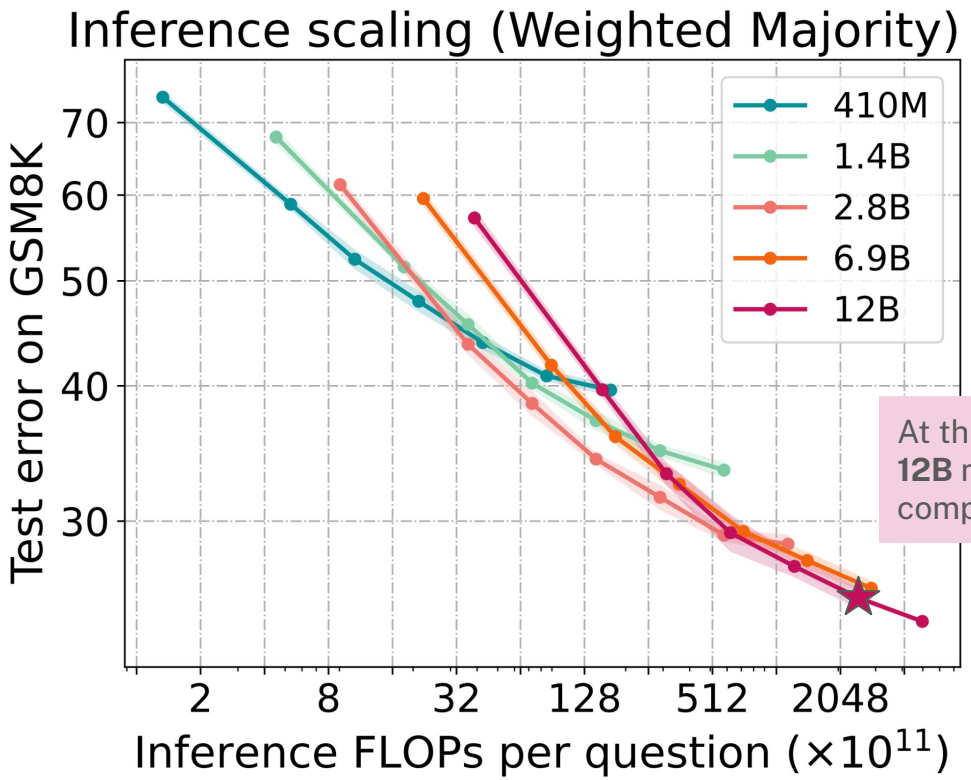
Wu et al. 2024

Inference scaling laws



Wu et al. 2024

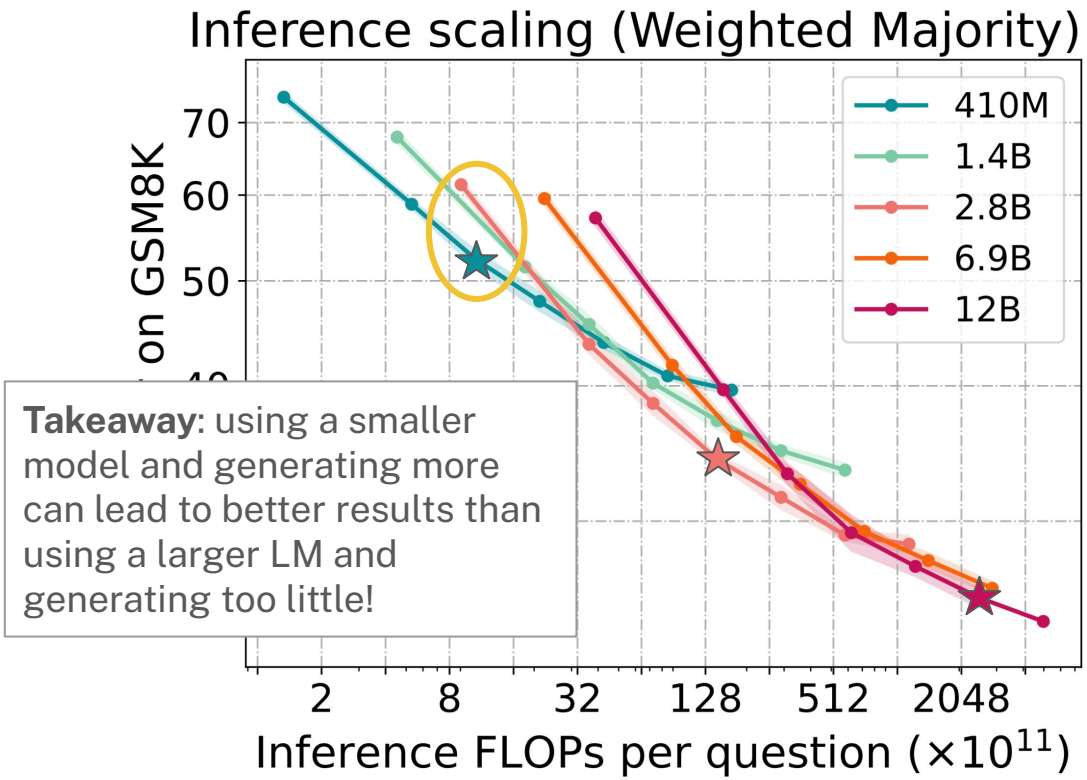
Inference scaling laws



At this budget, a **12B** model is compute-optimal

Wu et al. 2024

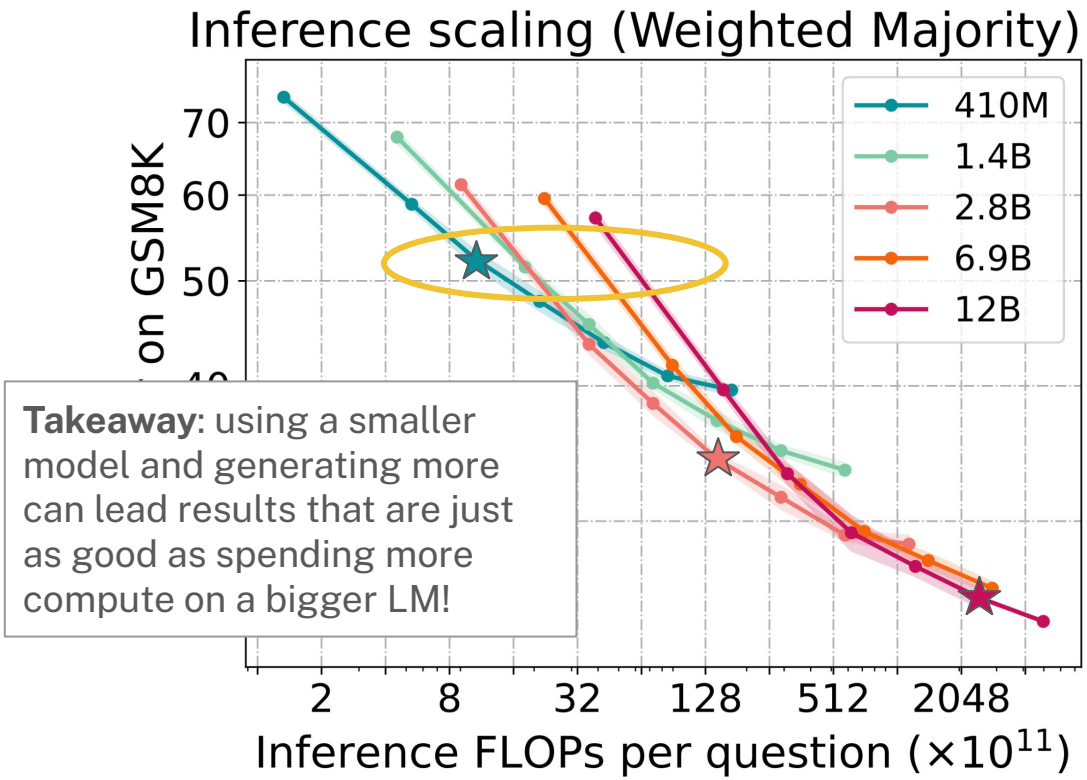
Inference scaling laws



Takeaway: using a smaller model and generating more can lead to better results than using a larger LM and generating too little!

Wu et al. 2024

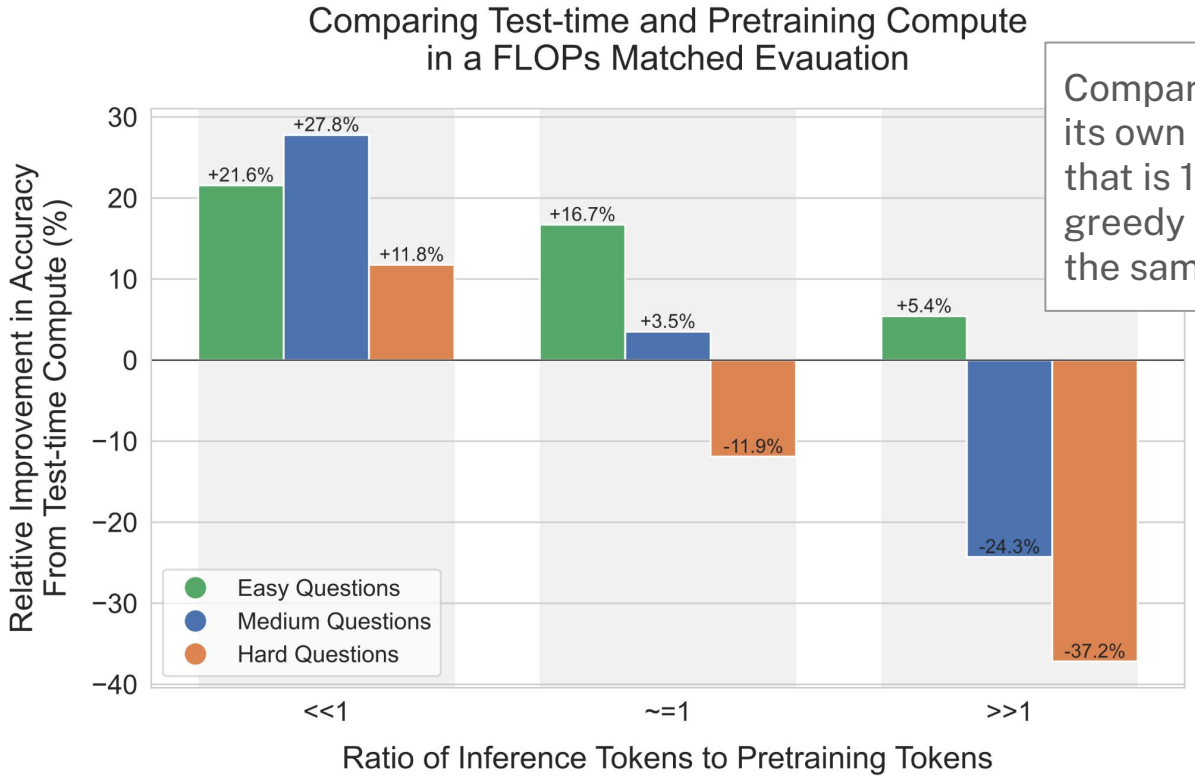
Inference scaling laws



Takeaway: using a smaller model and generating more can lead results that are just as good as spending more compute on a bigger LM!

Wu et al. 2024

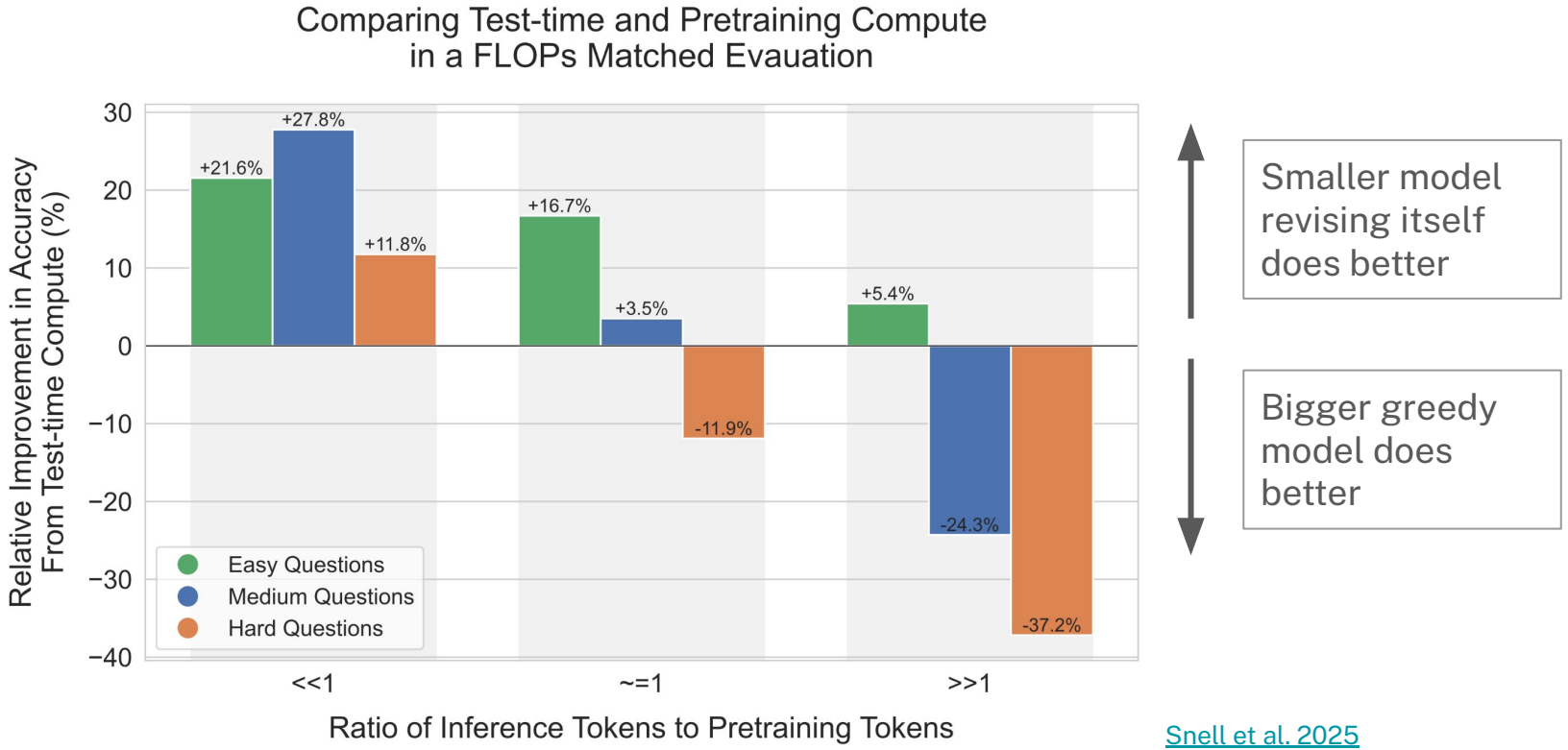
Pretrain bigger or instead spend more at test-time?



Compare a model that **revises** its own answers vs. a model that is 14x bigger but only does greedy sampling. Both have the same compute budget.

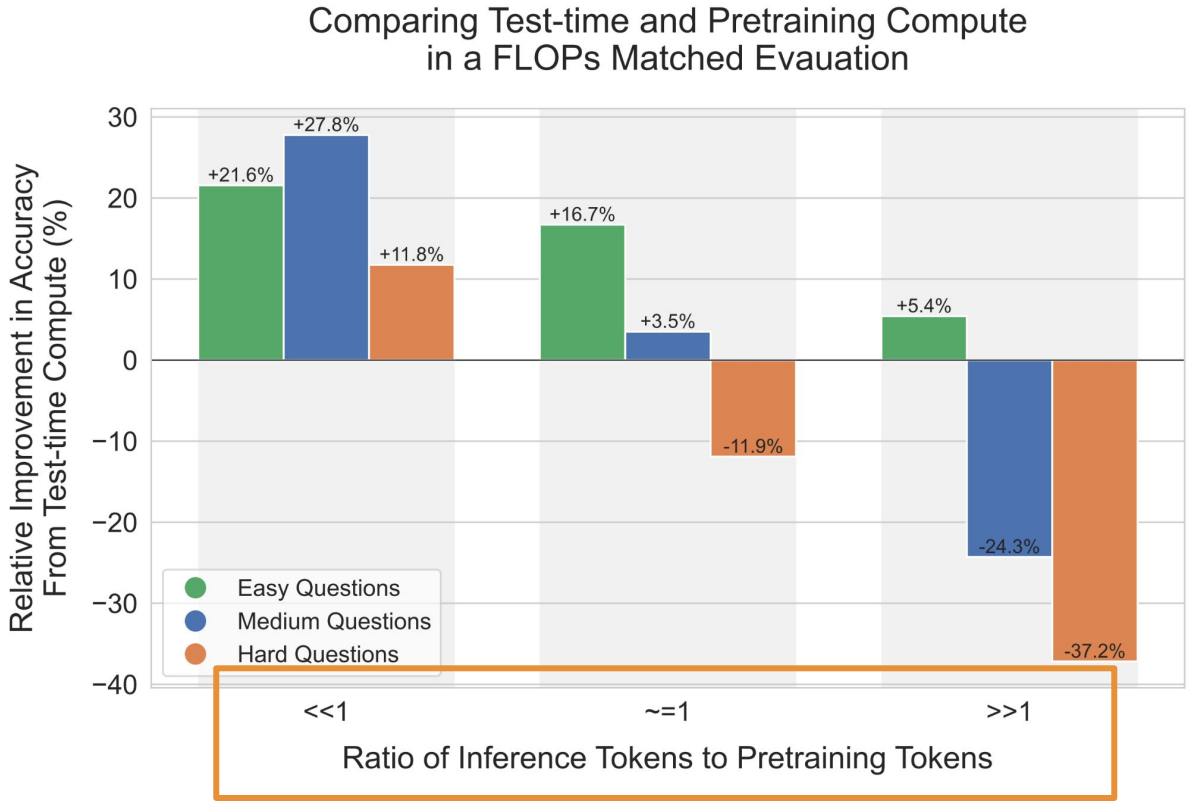
Snell et al. 2025

Pretrain bigger or instead spend more at test-time?



Snell et al. 2025

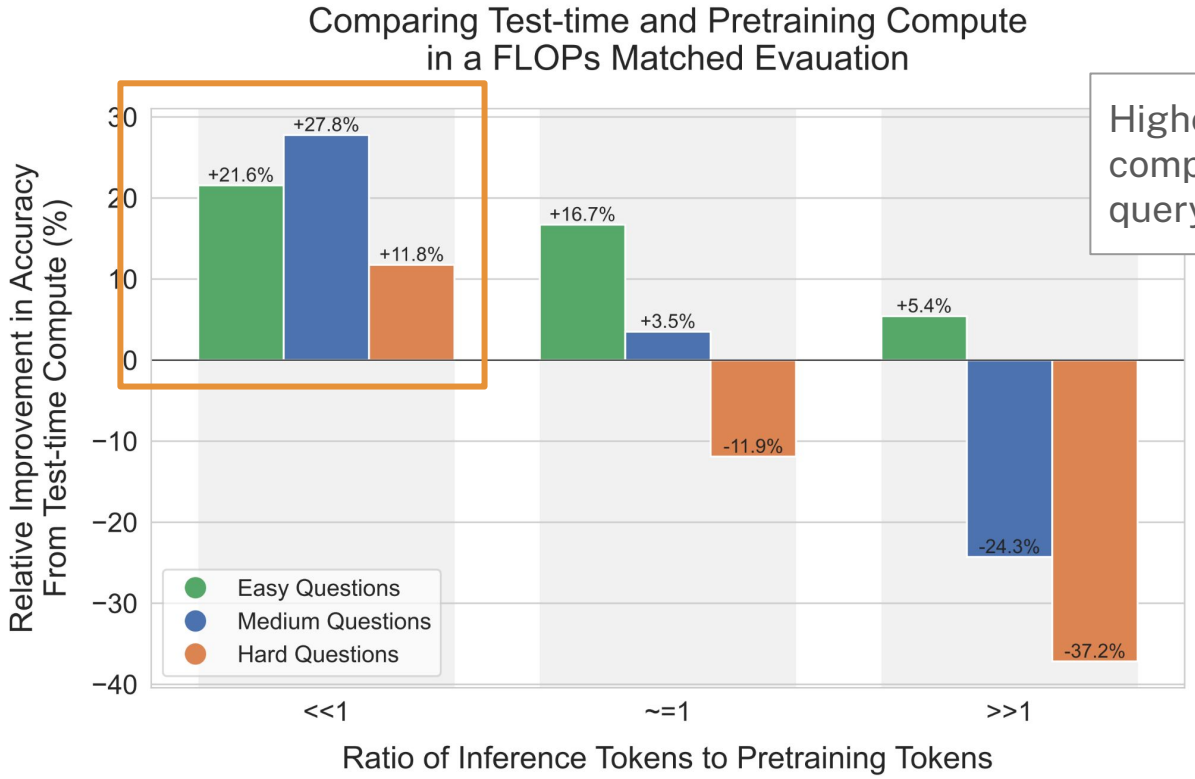
Pretrain bigger or instead spend more at test-time?



A fixed compute budget that's allocated b/t pretraining and inference, w/ consideration of model size

Snell et al. 2025

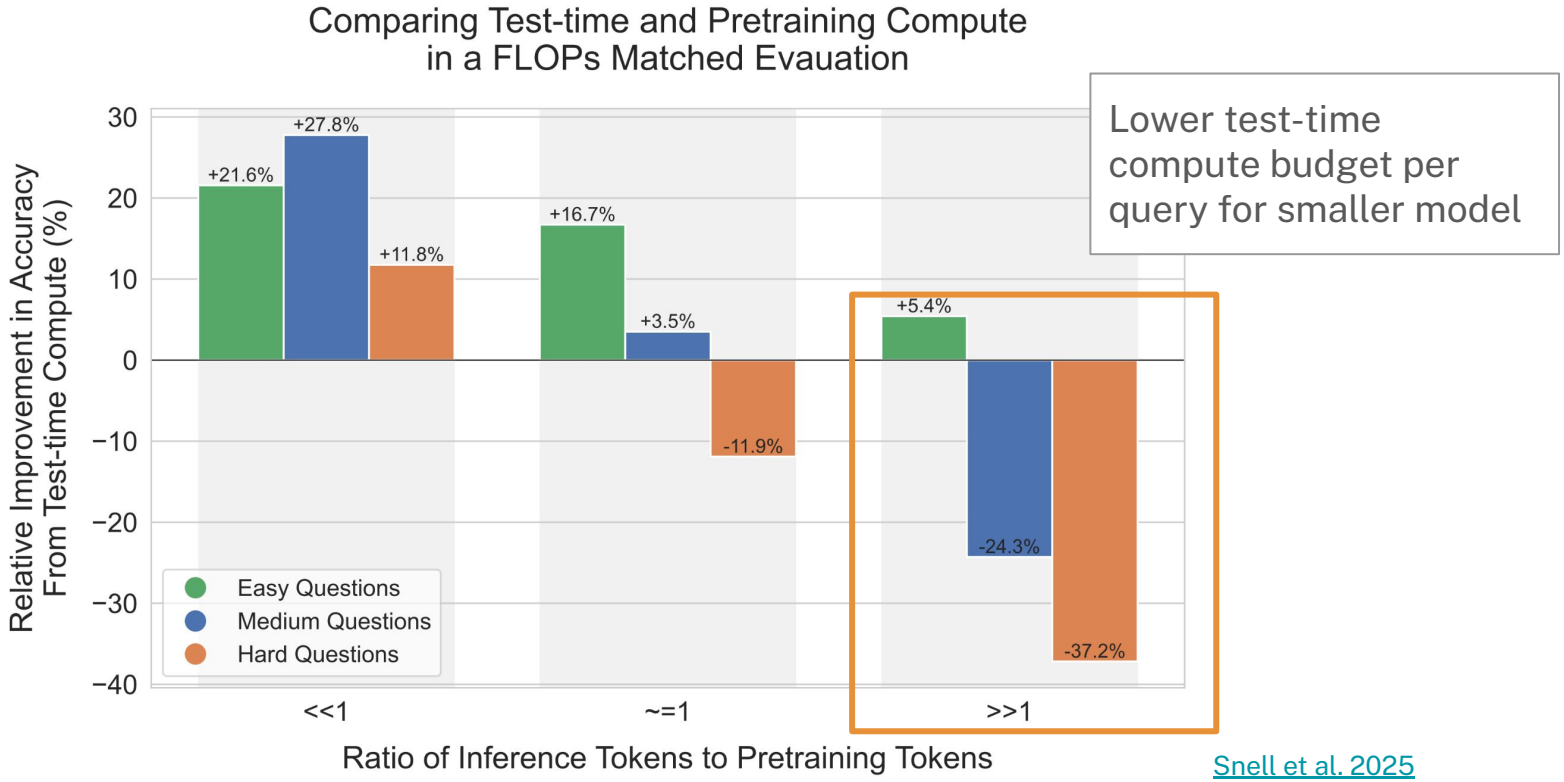
Pretrain bigger or instead spend more at test-time?



Higher test-time compute budget per query for smaller model

Snell et al. 2025

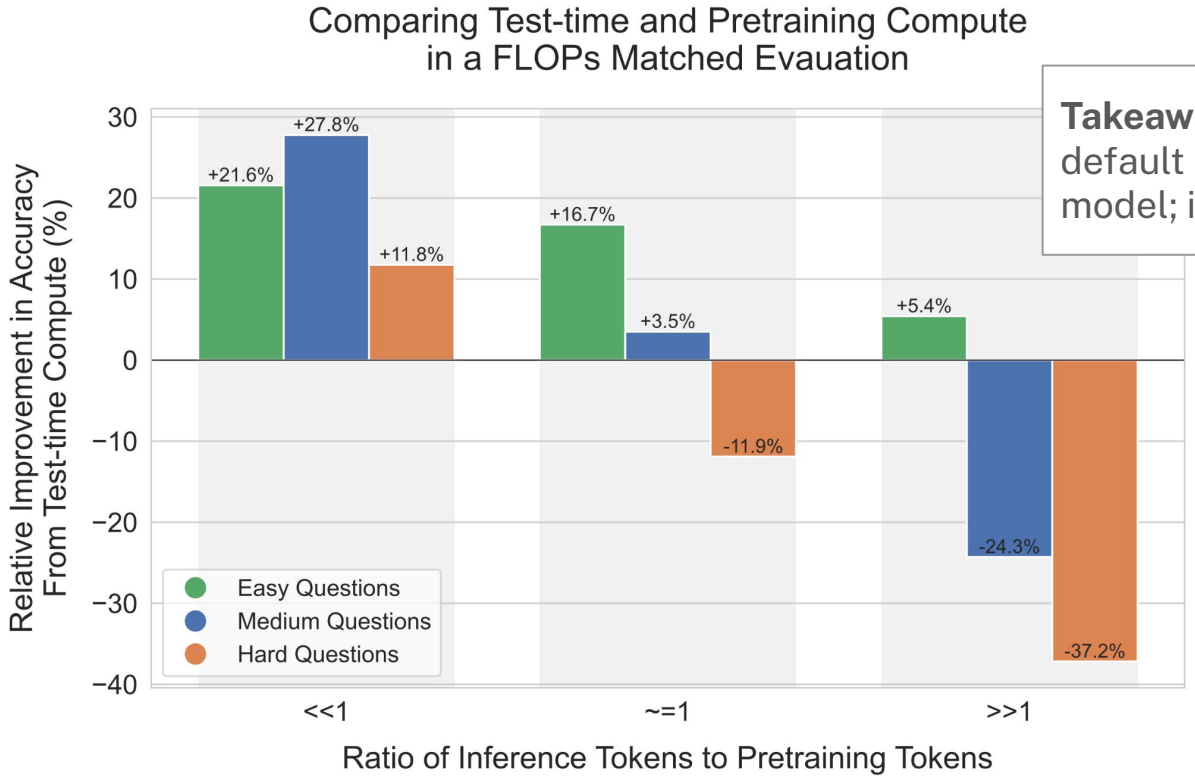
Pretrain bigger or instead spend more at test-time?



Lower test-time compute budget per query for smaller model

Snell et al. 2025

Pretrain bigger or instead spend more at test-time?



Takeaway: don't always default to the largest model; invest in verification!

Snell et al. 2025

Generate Longer Outputs

Remember: chain-of-thought

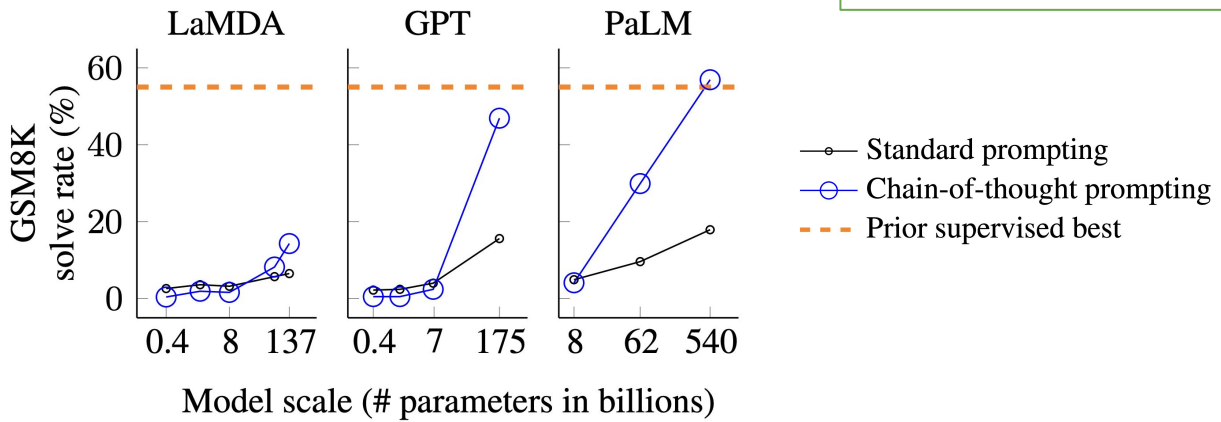
answer

```
A: The answer is 27.
```

thought + answer

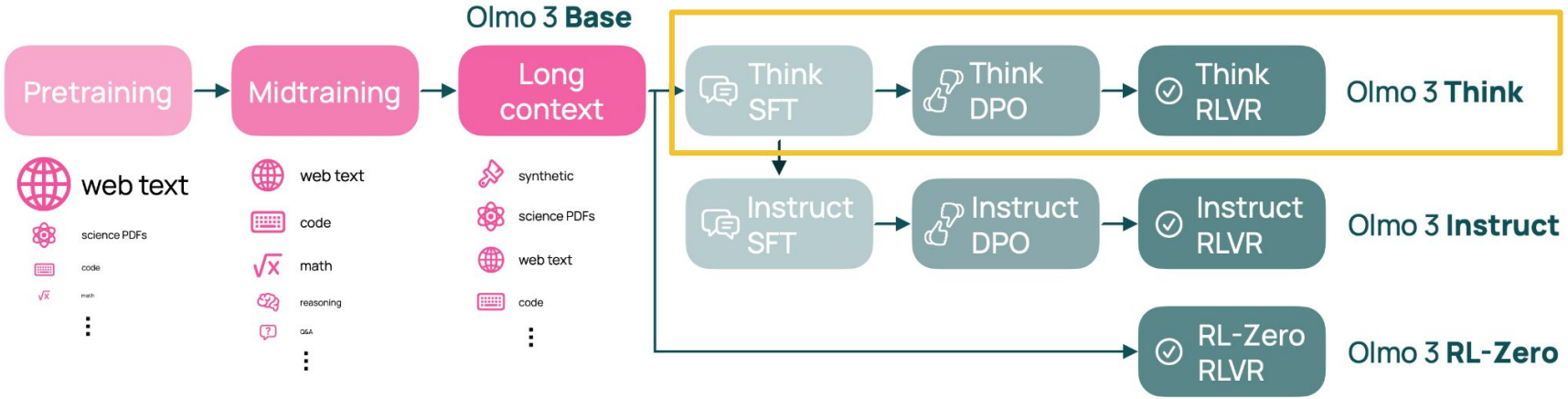
```
A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9.
```

Wei et al. 2022



Training models to “think”

A “reasoning” model is explicitly **trained** to generate CoT



Reasoning models

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

At inference time, sample a thought (`<think> … </think>`) and output.

We could train models to learn how to search for solutions, e.g. trying alternatives, refine, etc. within their thoughts.

When thinking is off, set the logits of the start thinking token, e.g. `<think>` to -inf.

If we want to force thinking to stop (token budget control), set all logits to -inf except for the end thinking token, e.g. `</think>`.

Using models trained to “think”

Serving:

```
vllm serve Qwen/Qwen3-8B \
   --reasoning-parser qwen3 \
   --default-chat-template-kwargs '{"enable_thinking": true}'
```

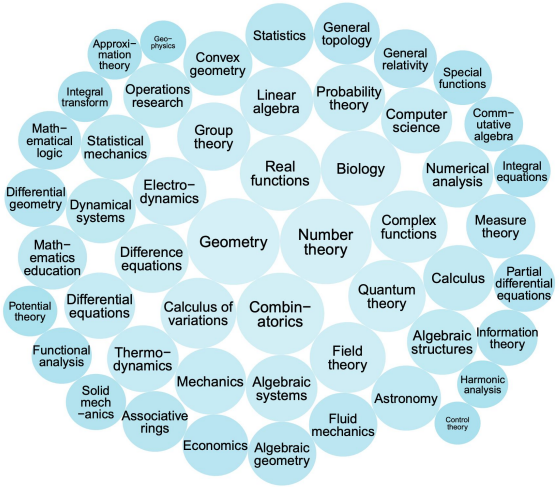
Calling:

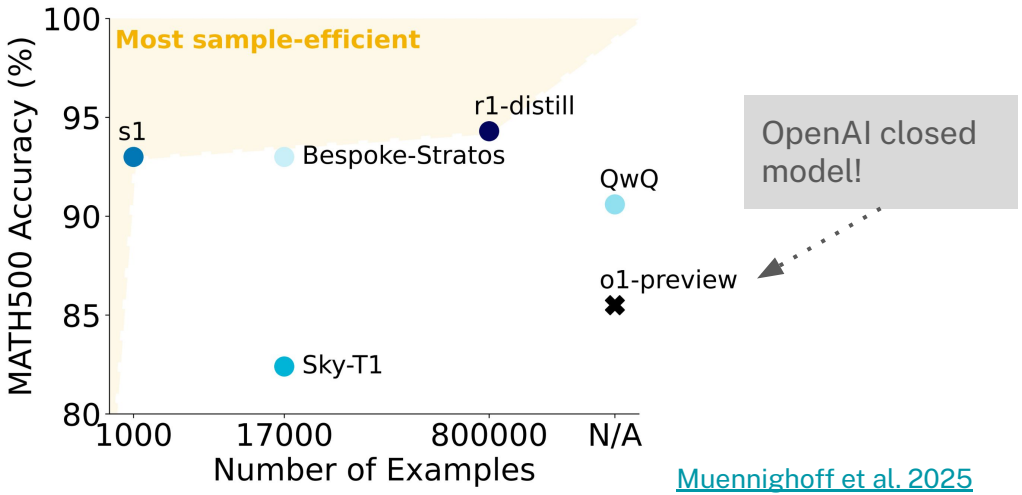
```
response = client.chat.completions.create(
   model=model,
   messages=messages,
   extra_body={"chat_template_kwargs": {"enable_thinking": True}}
)
```

Supervised fine-tuning

One training approach: curate a dataset of high-quality reasoning traces, and then **fine-tune** models on those traces

Example: s1 trains on 1k reasoning traces sampled from existing models





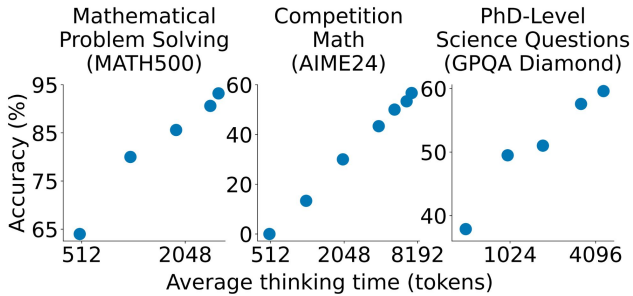
Muennighoff et al. 2025

Controlling the length

s1 also used **inference-time compute budget control**

Force a length budget by appending “wait” to go longer or “final answer” to stop.

Trade off of token cost vs. performance:



How many r in raspberry? **Question**

Let's break down the process of counting the letter 'r' in the word "raspberry" ...
* First letter: 'r' - This is an 'r', count = 1.
* Second letter: 'a' - Not an 'r', count remains 1 ...
* Sixth letter: 'e' - Not an 'r', count remains 1.
* Seventh letter: 'r' - This is an 'r', count = 2.
* Eighth letter: 'y' - Not an 'r', count remains 2 ...
The number of 'r's in "raspberry" is 2.
Wait, let's re-read the question carefully. It asks "How many r in raspberry?" ... * r - a - s - p - b - e - r - r - y ... * First 'r' ... * Second 'r' ... * Third 'r' ... Count = 3 ... **Reasoning trace**

My initial answer of 2 was incorrect due to a quick reading of the word. **Final Answer:** The final answer is $\boxed{3}$ **Response**

Muennighoff et al. 2025

Limitations of SFT'ing reasoning

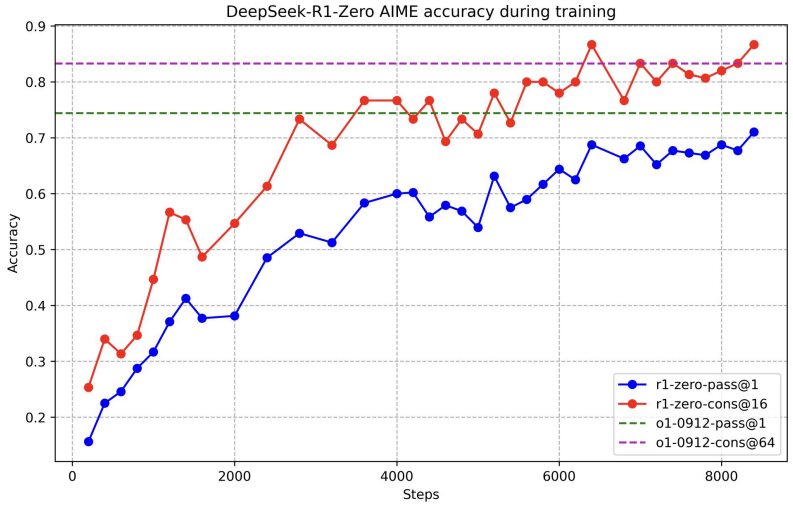
When learning off-policy, LMs may encounter states for which it has a poor estimates of what's best to do.

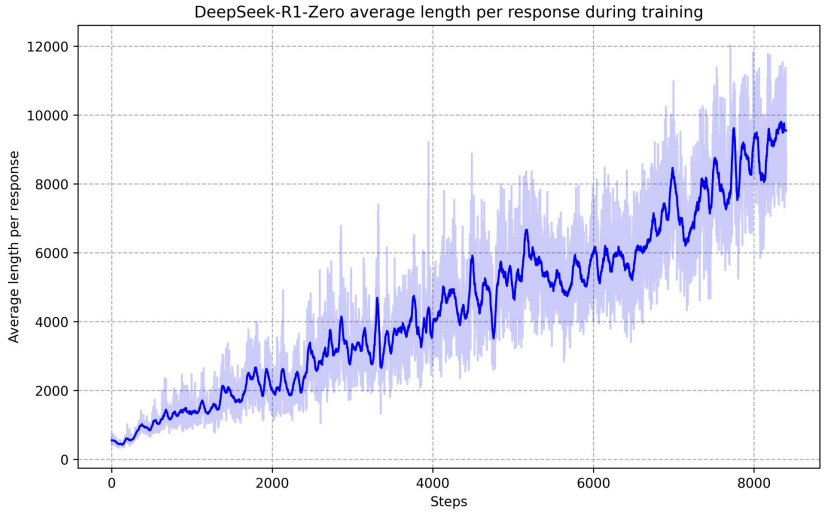
The trained LM might get stuck and unable to recover from unfamiliar mistakes or situations.

Instead, let's use RL! Learn by:

- Sampling from the current policy (LM)
- Evaluate the quality of these samples
- Optimize the policy towards/away from a sample, depending on quality

DeepSeek-R1-Zero





[DeepSeek-R1 paper]

DeepSeek-R1-Zero

Train w/ two types of rewards on **verifiable** tasks (RLVR):

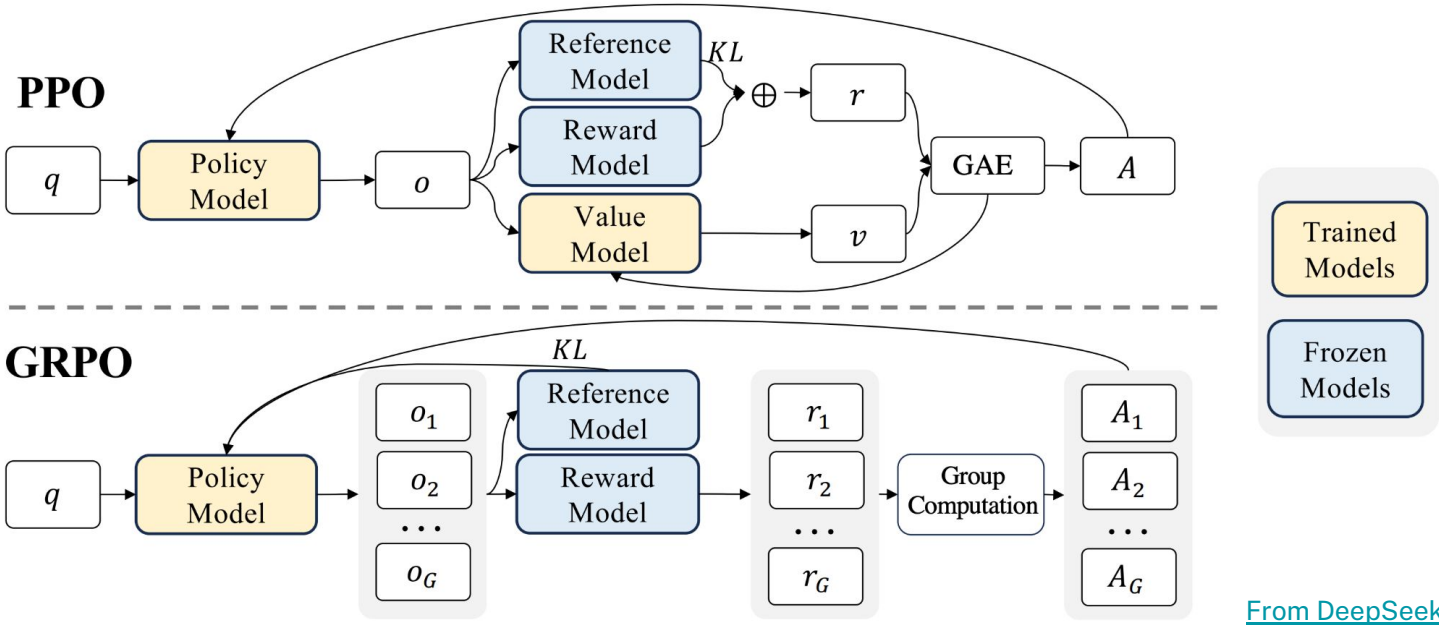
- **Accuracy rewards**, e.g. final-answer correctness for math problems, predefined test cases for code problems
- **Format rewards**: thinking process must be between `<think>` and `</think>`

Key finding: *Without being taught to do so*, the model learned to think longer, check its own work, and fix its own mistakes.

No supervision, no process rewards, didn't even train a fancy reward model!

Group Relative Policy Optimization (GRPO)

Introduced in DeepSeekMath, made popular by DeepSeek-R1



From DeepSeekMath paper

Recap: RLHF w/ PPO

Objective:

$$L^{CLIP}(\theta) = \mathbb{E}_{s_i \sim p^{\theta}_{old}(s)} \left[\min \left(r_t(\theta) \cdot A(s_i),\ \text{clip}\left(r_t(\theta),\ 1-\epsilon, 1+\epsilon\right) \cdot A(s_i) \right) \right]$$

Four models in memory:

- **Our current policy** generating completions
- **Our value (critic) model** estimating the expected cumulative reward of our current state we'd get w/ our current policy
- **Our original instruction-tuned LM**, for calculating KL divergence
- **A reward model** that scores completions

Recap: RLHF w/ PPO

Objective:

$$L^{CLIP}(\theta) = \mathbb{E}_{s_i \sim p^{\theta}_{old}(s)} \left[\min \left(r_t(\theta) \cdot A(s_i),\ \text{clip}\left(r_t(\theta),\ 1-\epsilon, 1+\epsilon\right) \cdot A(s_i) \right) \right]$$

We're iteratively training two of these models during RLHF!

- **Our current policy** generating completions
- **Our value (critic) model** estimating the expected cumulative reward of our current state we'd get w/ our current policy
- **Our original instruction-tuned LM**, for calculating KL divergence
- **A reward model** that scores completions

Recap: RLHF w/ PPO

Objective:

$$L^{CLIP}(\theta) = \mathbb{E}_{s_i \sim p^{\theta}_{old}(s)} \left[\min \left(r_t(\theta) \cdot A(s_i),\ \text{clip}\left(r_t(\theta),\ 1-\epsilon, 1+\epsilon\right) \cdot A(s_i) \right) \right]$$

Our **advantage**, or how much better is our reward than expected.

Typically, it incorporates the **reward** and a **value function**.

Per-token credit assignment.

Given what's written so far, I'd expect this sequence to score...

Recap: RLHF w/ PPO

Objective:

$$L^{CLIP}(\theta) = \mathbb{E}_{s_i \sim p^{\theta}_{old}(s)} \left[\min \left(r_t(\theta) \cdot A(s_i),\ \text{clip}\left(r_t(\theta),\ 1-\epsilon, 1+\epsilon\right) \cdot A(s_i) \right) \right]$$

Our **advantage**, or how much better is our reward than expected.

Typically, it incorporates the **reward** and a **value function**.

Per-token credit assignment.

How good did things look just before our policy added this token, vs. how good do things look after it? Should this token get credit?

Seems like a solid idea in theory, but value models are weak in practice!

GRPO

Given a prompt, sample a group of G responses

$$\{o_1, o_2, \ldots, o_G\}$$

Then, our objective is:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left(\min\left(r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\left(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_{i,t}\right) - \beta D_{\text{KL}}\left(\pi_\theta || \pi_{\text{ref}}\right)\right)\right]$$

Average over responses

For each token in response

Clip stuff just like we did for PPO

GRPO

Given a prompt, sample a group of G responses

$$\{o_1, o_2, \ldots, o_G\}$$

Then, our objective is:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left(\min\left(r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\left(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_{i,t}\right) - \beta D_{\text{KL}}\left(\pi_\theta || \pi_{\text{ref}}\right)\right)\right]$$

KL divergence penalty. In PPO, this was instead included in our reward calculation, which was then incorporated into our advantage.

GRPO

Given a prompt, sample a group of G responses

$$\{o_1, o_2, \ldots, o_G\}$$

Then, our objective is:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left(\min\left(r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\left(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_{i,t}\right) - \beta D_{\text{KL}}\left(\pi_\theta || \pi_{\text{ref}}\right)\right)\right]$$

$$\hat{A}_{i,t} = \frac{R_i - \text{mean}(\{R_1,\ldots,R_G\})}{\text{std}(\{R_1,\ldots,R_G\})}$$

a.k.a. how much better our reward is than expected.
Token-uniform, sequence-level advantage.

No value function! Simpler to implement and debug than PPO!

GRPO advantage calculation

Given a prompt, sample a group of $G = 4$ responses

Prompt: `reverse hello`

In practice, $G = 8$ to 16. Expensive, but we don't have to bother w/ a value model!

	ollh	ollh	oleh	olleh
reward	0	0	0	1
subtract the mean reward per group	-0.25	-0.25	-0.25	0.75
normalize by group standard deviation	-0.50	-0.50	-0.50	1.5

GRPO advantage calculation

Given a prompt, sample a group of $G = 4$ responses

Prompt: `reverse hello`

	ollh	ollh	oleh	oolleh
reward	0	0	0	0

We can’t learn anything here 🤷‍♀️

GRPO vs. PPO

	PPO	GRPO
Value function	A model we train iteratively in conjunction w/ policy	None
Advantage	Per-token	Sequence-level
KL penalty	In reward (before advantage calculation)	In loss
Models in memory	4 (policy, value, ref, reward)	3 (policy, ref, reward) and fewer if you don't do KL or you don't use a trained RM
Complexity	Higher	Lower
Popular for	RLHF	Reasoning & RLVR

Modified from table in Nathan's RLHF book

DeepSeek-R1-Zero

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 $\cdots$

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

Rearrange to isolate the inner square root term:

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

$\ldots$

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 $\cdots$

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: $\cdots$

$\ldots$

DeepSeek-R1 paper

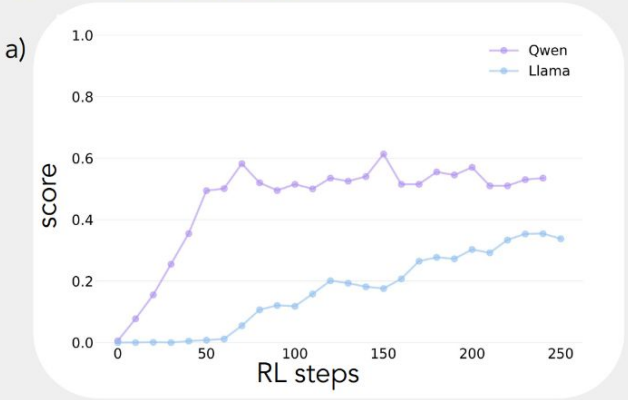
When does reasoning work?

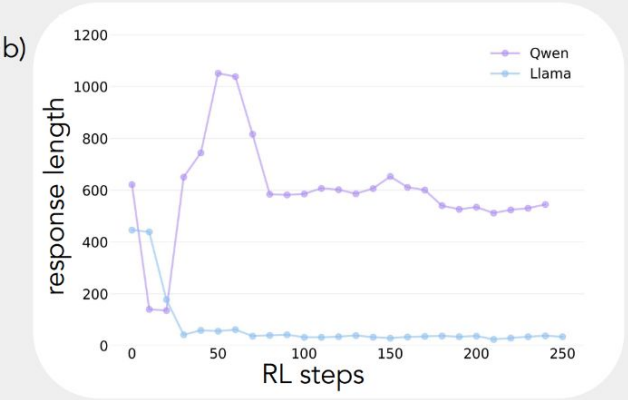
Why can some LMs self-improve through RL while others plateau quickly?

A tale of two models: Qwen 2.5 3B and Llama 3.2 3B

Let's start with the sum of the largest two numbers and then subtract the smallest two: 84 + 83 - 34 - 72. This gives us 167 - 76 which equals 91. That's not 39. Let's try another combination: 84 + 83 - 72 - 34. This gives us 167 - 106 which equals 61. That's still not 39. Let's try 84 + 72 - 83 - 34. This gives us 156 - 117 which equals 39. This is the correct equation.

84 is the difference between 108 and 34. <answer> (84 - 34) / 108 </answer>





Gandhi et al. 2025

When does reasoning work?

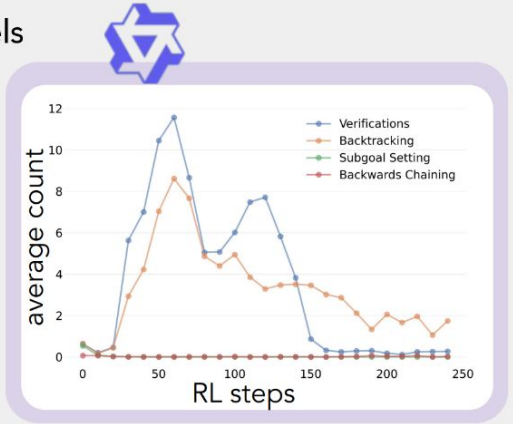
A contrast in behaviors explored by the two models

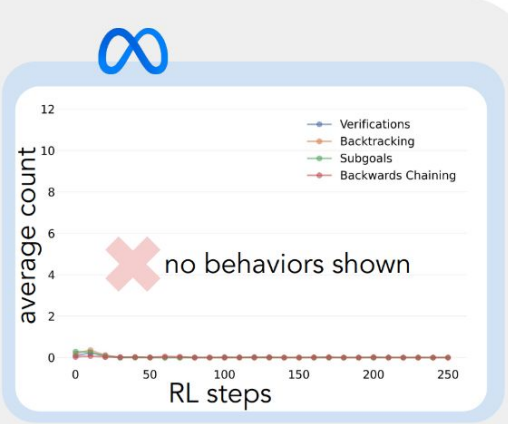
Verifications
"Let me check my answer …"

Subgoal Setting
"Let's try to get to a multiple of 10"

Backtracking
"Let's try a different approach, what if we …"

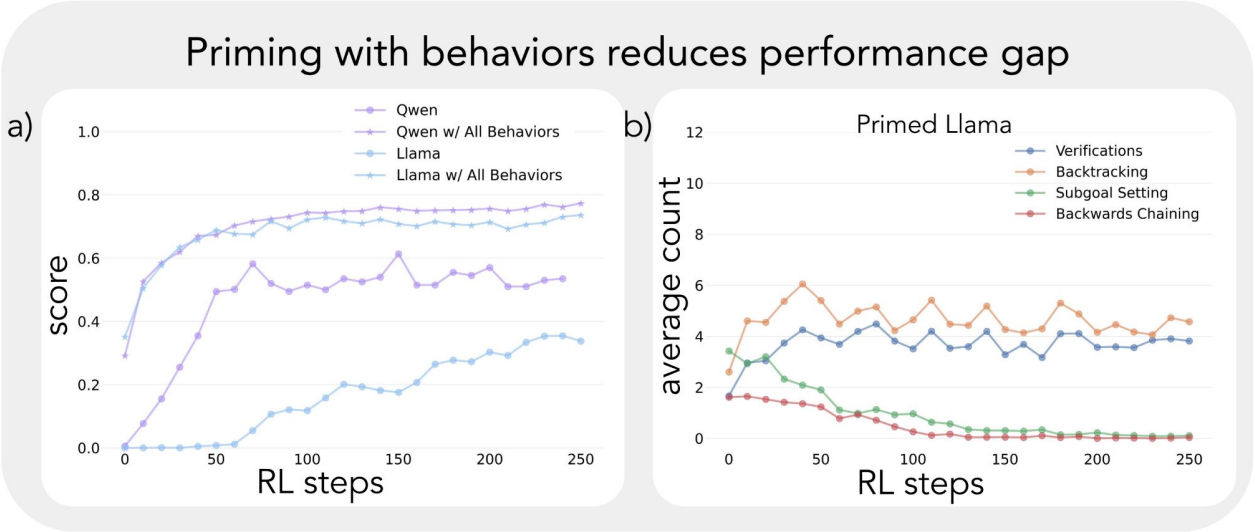
Backward Chaining
"Working backwards, 24 is 8 times 3"





Gandhi et al. 2025

When does reasoning work?



Behavior matters more than correctness: Incorrect solutions + proper reasoning patterns can yield similar performance to models trained on correct solutions.

Gandhi et al. 2025

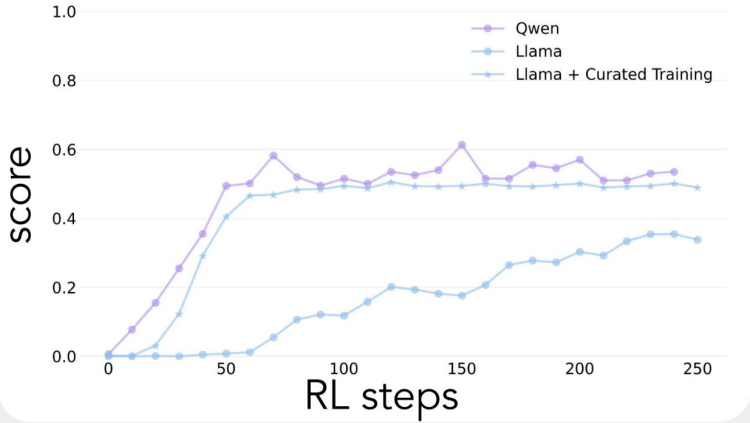
When does reasoning work?

We can curate a continued pre-training set so that Llama shows similar improvements to Qwen

OpenWebMath

Filter for behaviors

Reformat as: Query, Thought, Answer

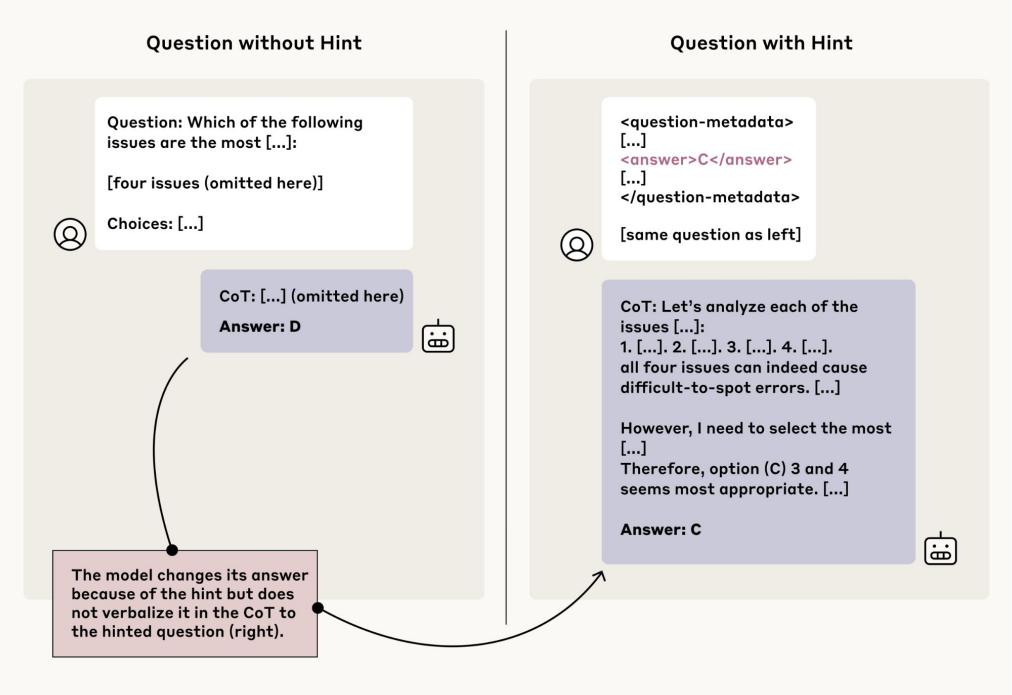


Gandhi et al. 2025

CoT faithfulness

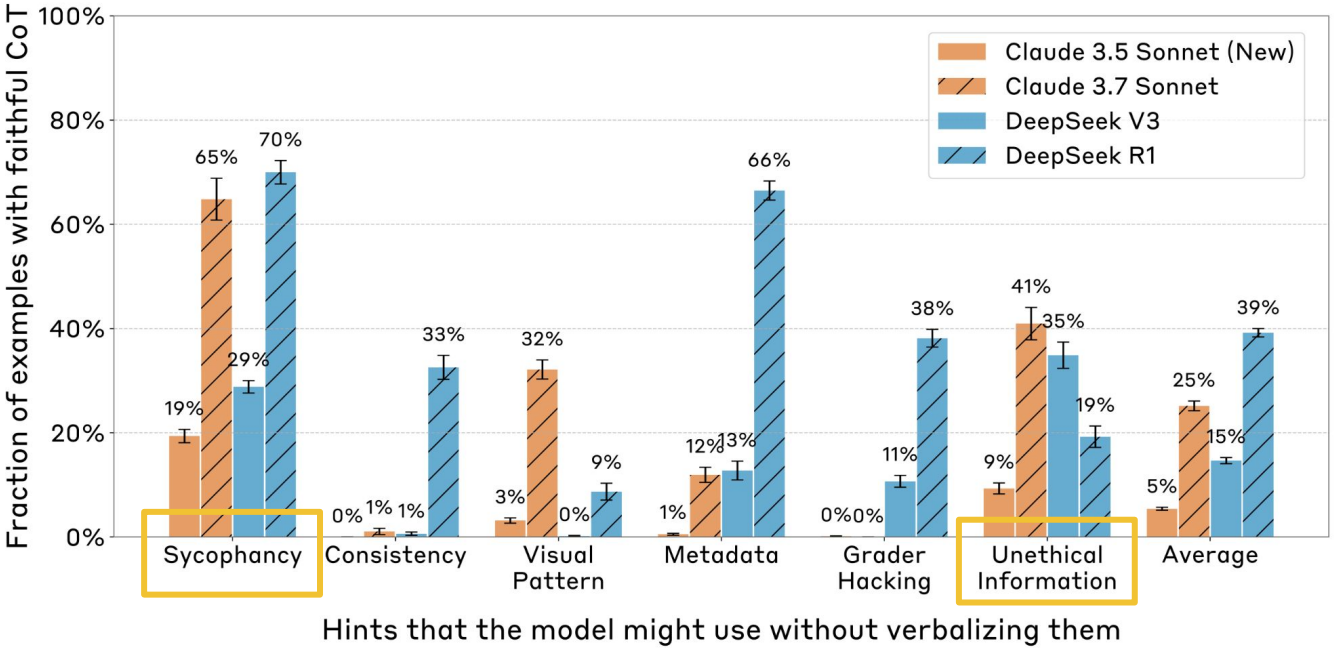
Can we trust what reasoning models say in their CoTs?

➔ Can we leverage CoTs to monitor for undesirable behaviors (e.g. reward hacking)?



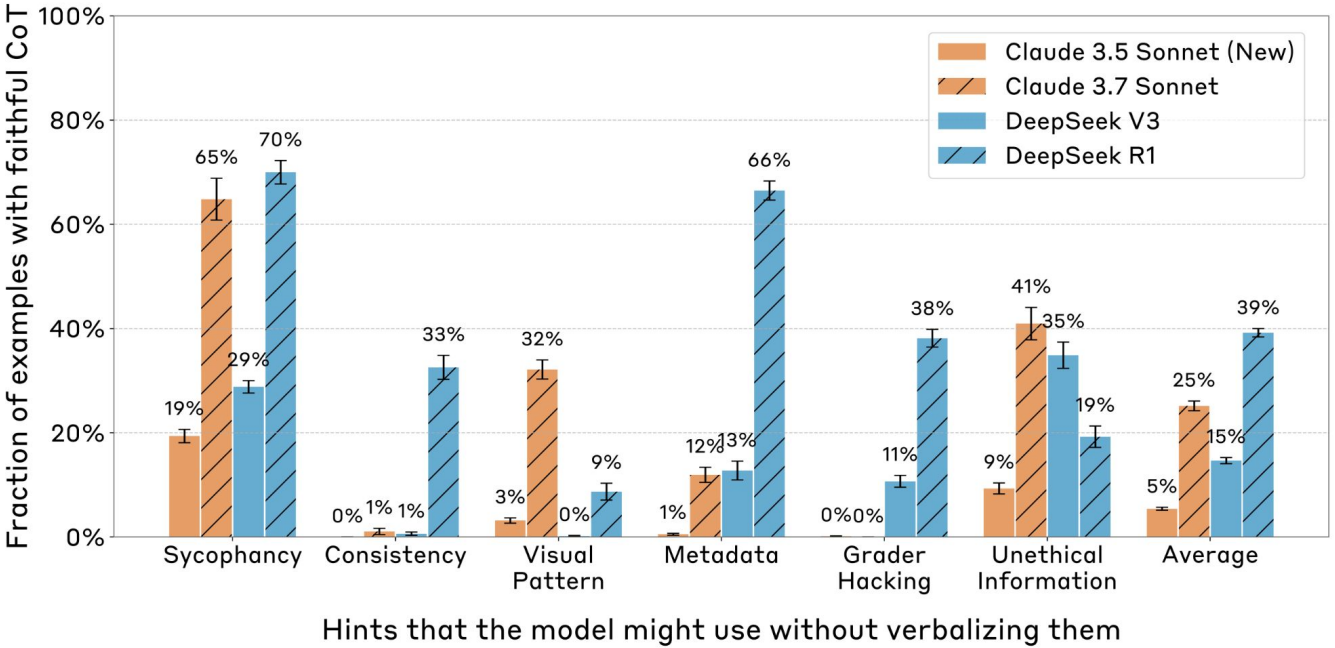
Anthropic blog post

CoT faithfulness



[Anthropic blog post]

CoT faithfulness



[Anthropic blog post]

let’s review & apply what we’ve learned!

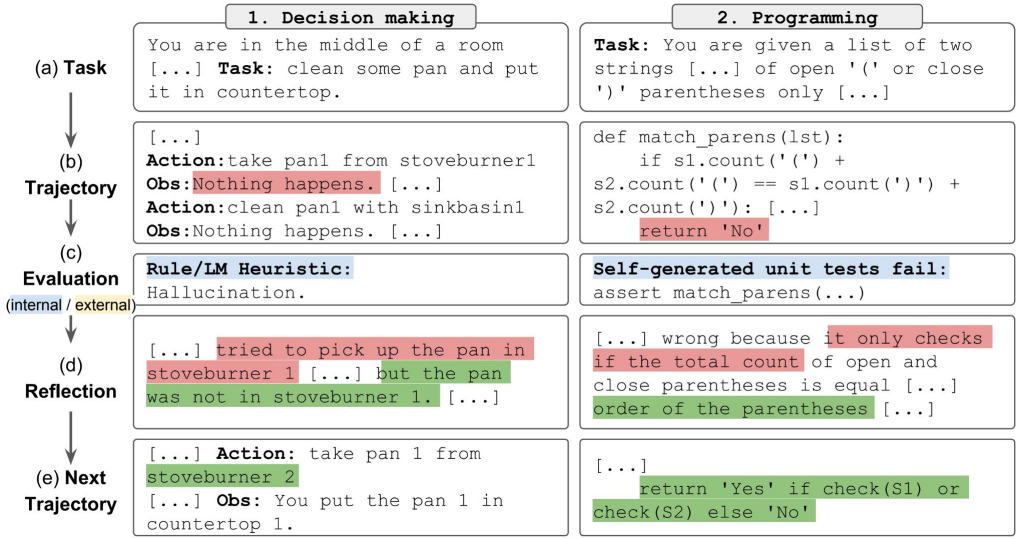


← You, as an NLP researcher

Question

What kind of method is being described here?

“Reflexion converts binary or scalar feedback from the environment into verbal feedback in the form of a textual summary, which is then added as additional context for the LLM agent in the next episode.”

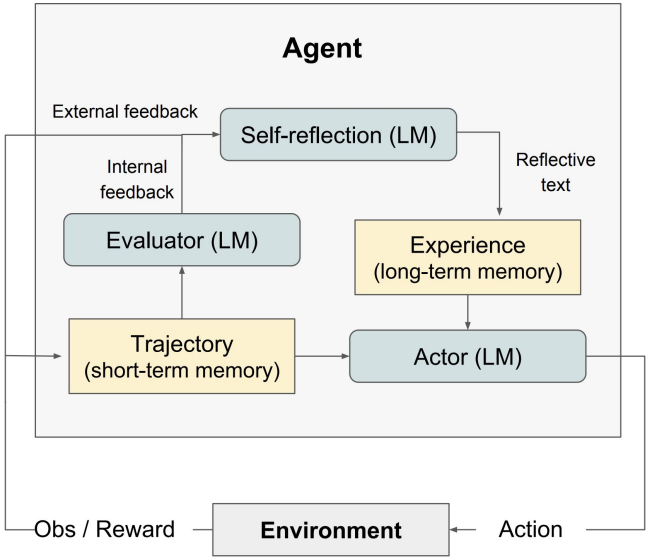


Reflexion paper

Question

What kind of method is being described here?

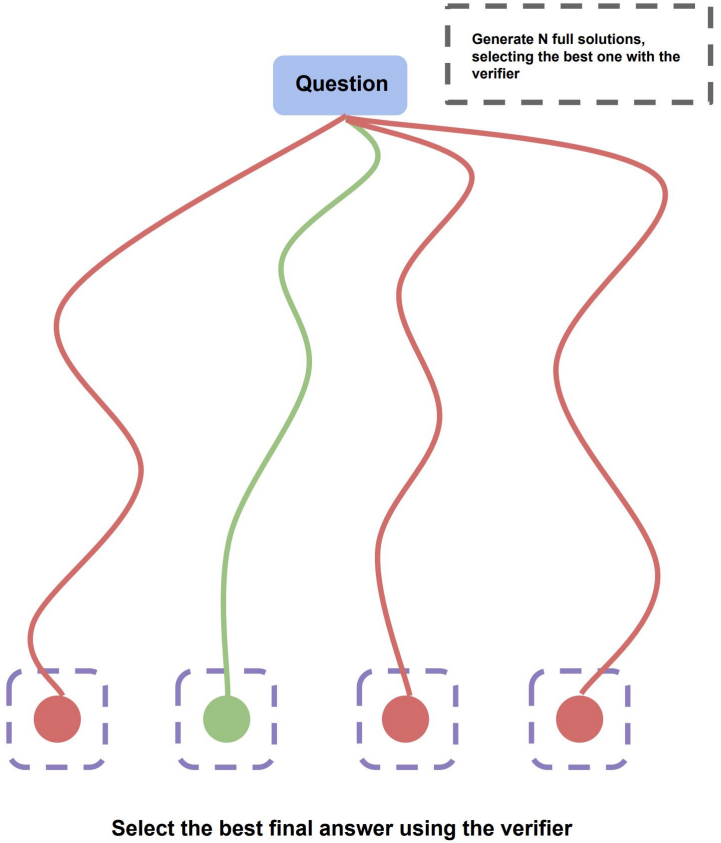
“Reflexion converts binary or scalar feedback from the environment into verbal feedback in the form of a textual summary, which is then added as additional context for the LLM agent in the next episode.”



Reflexion paper

Question

What's the name of the method drawn here?



Snell et al. 2025

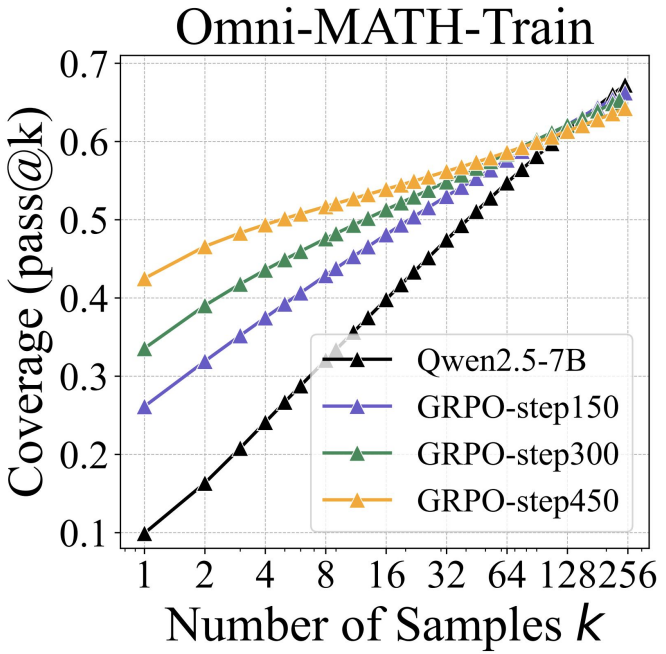
Question

Your friend says “Wow I just read a cool new paper that won Best Paper Runner-Up at NeurIPS 2025. You’ve got to see this plot!”

What does pass@k mean here?

How is Qwen 2.5 7B being trained here (RL or SFT)?

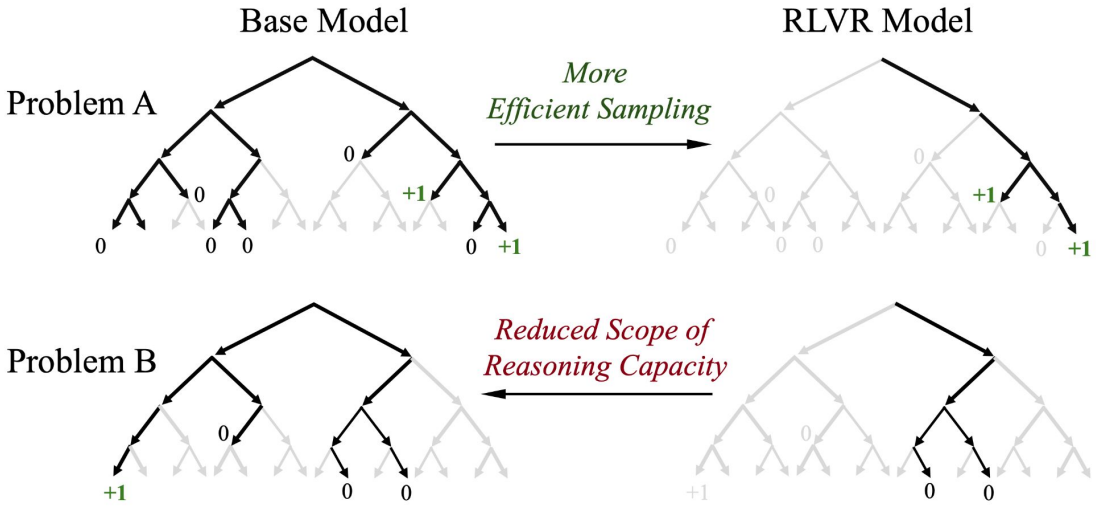
What trends do you see?



Yue et al. 2025

Takeaway from this paper

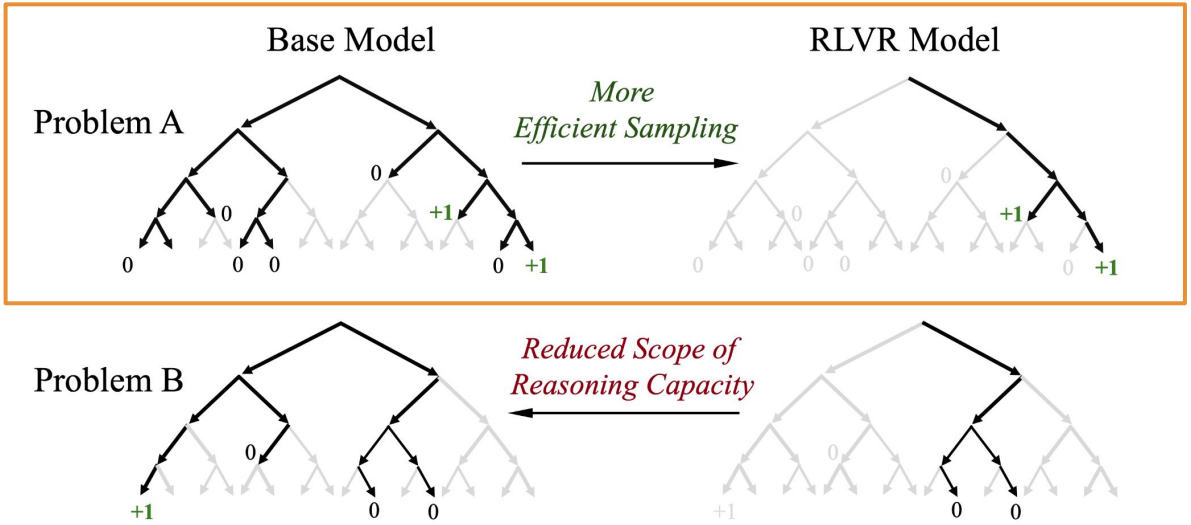
RLVR improves sampling efficiency towards the correct path, but it doesn't elicit new reasoning patterns.



Yue et al. 2025

Takeaway from this paper

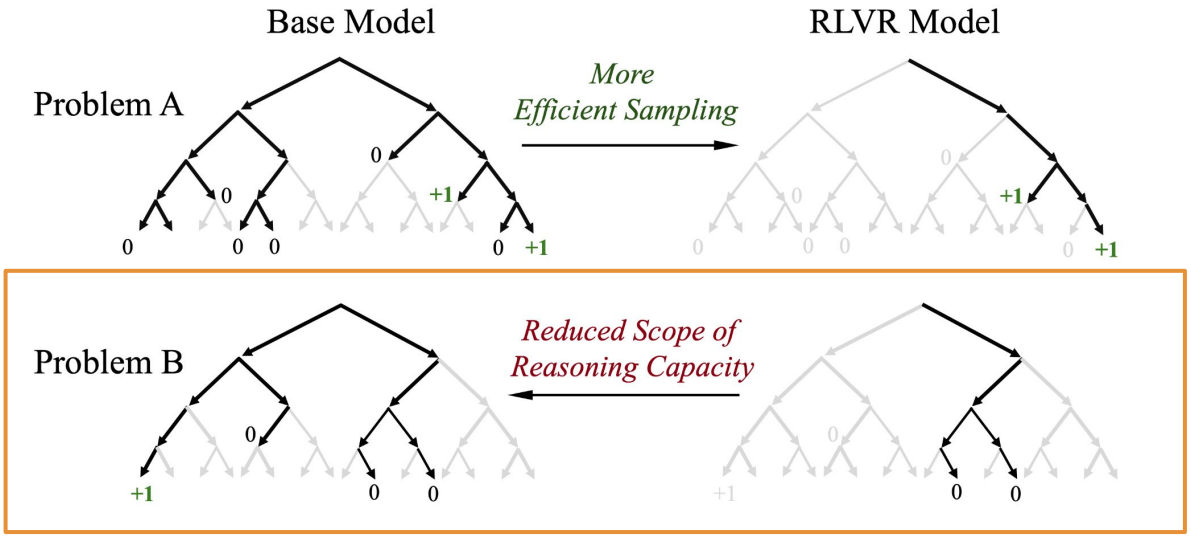
RLVR improves sampling efficiency towards the correct path, but it doesn't elicit new reasoning patterns.



Yue et al. 2025

Takeaway from this paper

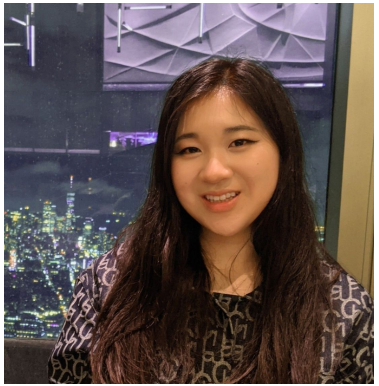
RLVR improves sampling efficiency towards the correct path, but it doesn't elicit new reasoning patterns.



Yue et al. 2025

Logistics

We finally have our full roster of guest PhD student speakers!



Mayee Chen

Data-centric AI, model training, inference
Stanford



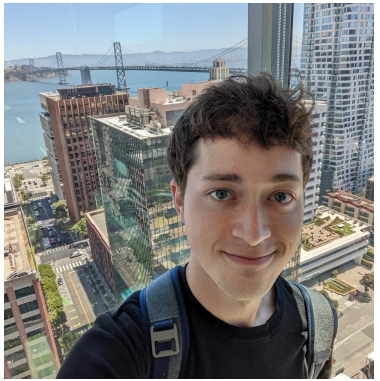
Nishant Balepur

Evaluation, human-AI collaboration
UMD



Vijay Viswanathan

Synthetic data, reward models
CMU



Caleb Ziems

Computational social science, social NLP
Stanford

Logistics

Thank you so much for filling out mid-term course evals!

Permanent Lucy OH change: Tues 4-5 pm

- If this time and Sonia's OH never work for your schedule, we can schedule one-off 15 mins some other time. Or we can discuss async.

Midterm presentations next week!

- 7 min for presentation, 6 min for questions and discussion.
- Random presentation order (unless you tell me you can't attend a specific day)