

# Policy gradient algorithms (cont'd) + Test-time scaling

**CS 6804: Frontier AI Systems**  
*Spring 2026*

<https://tuvllms.github.io/ai-seminar-spring-2026/>

**Tu Vu**



# Logistics

- Homework assignments
  - Homework 1, due 3/10
    - 5% extra credits each

# Policy Gradient review

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{y \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(y|x) R(x, y)]$$

A sequence probability factorizes:

$$\log \pi_{\theta}(y|x) = \sum_t \log \pi_{\theta}(y_t|x, y_{<t}).$$

Each sampled action contributes a gradient, scaled by how good the outcome was

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[ \sum_t \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) R \right].$$

# Advantage form (most important)

In modern algorithms:

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[ \sum_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A_t \right]$$

$A_t$  (or  $\Psi_t$ )

Learning signal.

- Positive  $\rightarrow$  action better than expected  $\rightarrow$  reinforce.
- Negative  $\rightarrow$  worse  $\rightarrow$  suppress.
- Zero  $\rightarrow$  no update.

**Policy gradient updates the model by increasing the log-probability of actions that produced better-than-expected outcomes and decreasing those that produced worse outcomes.**

## KL penalty per token (reward shaping)

$$R_{\text{shaped}} = R_{\text{model}} - \beta \sum_t \text{KL}_t$$

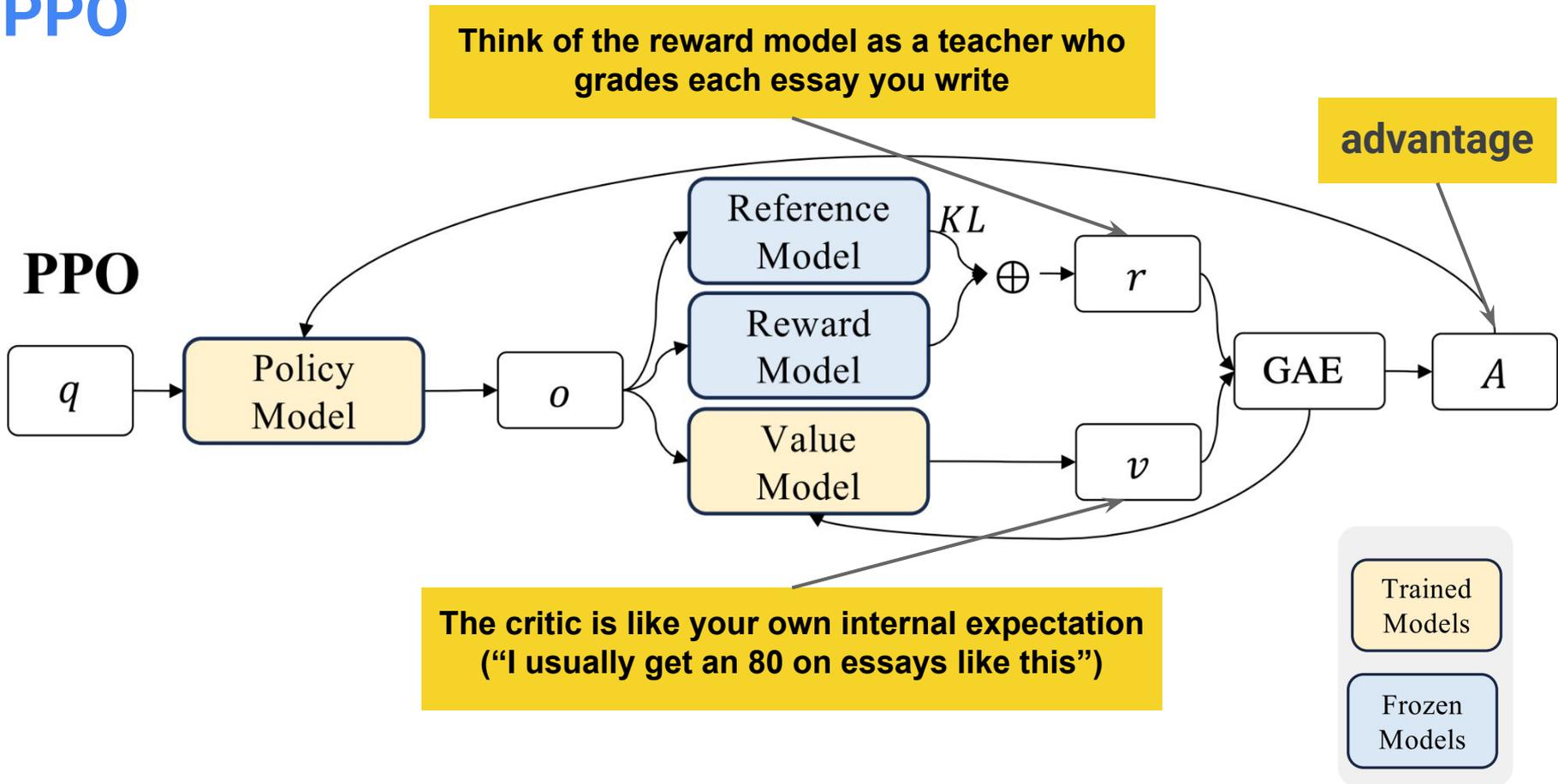
So the KL is computed **per token**, then summed over the sequence.

# Policy Gradient Algorithms (cont'd)

- REINFORCE
- RLOO (REINFORCE Leave One Out)
- PPO (Proximal Policy Optimization)
- **GRPO (Group Relative Policy Optimization)** 

# PPO

## PPO



## Monte Carlo advantage

$$A_t = G_t - V(s_t)$$

# 1) What the value model learns

Value model predicts a scalar

$$V(s_t) \approx \mathbb{E}[G_t | s_t]$$

where the "state" is the prefix:

$$s_t = (x, a_1, \dots, a_{t-1})$$

So input is **prompt + previously generated tokens**.

Output is **one number per token position**.

## 2) Collect training data (rollouts)

Pick a prompt:

x: Translate to French: "Hello world"

Sample a completion with the current policy:

y: "Bonjour le monde"

Tokens:  $a_1 = \text{Bonjour}$ ,  $a_2 = \text{le}$ ,  $a_3 = \text{monde}$ .

Now you have prefix states:

- $s_1 = \text{prompt}$
- $s_2 = \text{prompt} + \text{"Bonjour"}$
- $s_3 = \text{prompt} + \text{"Bonjour le"}$

These are your **training inputs** to the value model.

$$R = 0.9$$

Define rewards per step (simplest RLHF case, no KL):

- $r_1 = 0$
- $r_2 = 0$
- $r_3 = 0.9$  (terminal)

(If you include KL, then  $r_t$  also has  $-\beta \text{KL}_t$  terms.)

So the supervised dataset looks like:

input prefix $s_t$	label $G_t$
prompt	0.9
prompt + Bonjour	0.9
prompt + Bonjour le	0.9

$$R = 0.9$$

Define rewards per step (simplest RLHF case, no KL):

- $r_1 = 0$
- $r_2 = 0$
- $r_3 = 0.9$  (terminal)

(If you include KL, then  $r_t$  also has  $-\beta \text{KL}_t$  terms.)

So the supervised dataset looks like:

input prefix $s_t$	label $G_t$
prompt	0.9
prompt + Bonjour	0.9
prompt + Bonjour le	0.9

- TD residual:

$$\delta_t = r_t + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t).$$

- GAE:

$$A_t^{GAE} = \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots$$

- TD residual:

$$\delta_t = r_t + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t).$$

- GAE:

$$A_t^{GAE} = \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots$$

## Monte Carlo target

$$L_{\text{value}}(\theta) = \frac{1}{N} \sum_t (V_{\theta}(s_t) - G_t)^2$$

## GAE target

$$L_{\text{value}}(\theta) = \frac{1}{N} \sum_t (V_{\theta}(s_t) - (A_t + V_{\text{old}}(s_t)))^2$$

# Reward model vs. Value model

## 1. The reward model learns a function of full sequences

Reward model:

$$R(x, y)$$

Input:

(prompt, full completion)

Example:

input	reward
(Hello, "Bonjour le monde")	0.9
(Hello, "Salut le chien")	0.1

It never sees prefixes.

# Reward model vs. Value model (cont'd)

## 2. The value model sees prefixes

Value model input:

$$s_t = (x, a_1, \dots, a_{t-1})$$

Examples during training:

prefix	label $G_t$
prompt	0.9
prompt + Bonjour	0.9
prompt + Bonjour le	0.9

But across many trajectories the same prefix can lead to **different final rewards**.

Example dataset:

# Reward model vs. Value model (cont'd)

prefix	completion	reward
Bonjour	Bonjour le monde	0.9
Bonjour	Bonjour le chien mange une voiture	0.1

So the value model learns

$$V(\text{Bonjour}) \approx 0.5$$

which is the **expected reward of continuations under the policy.**

### 3. The reward model cannot compute this expectation

The reward model only knows

$$R(x, y)$$

for a **specific finished sequence**.

It does not model

$$\mathbb{E}_\pi[R(x, y) \mid s_t]$$

which is what the value function must approximate.

---

### 4. The value function depends on the policy

This expectation depends on the **current policy**.

As the policy improves:

prefix → better completions

so

expected reward increases

The value model must be **updated continuously**.

The reward model stays fixed.

## 5. Another practical reason

Reward models are usually:

- trained with preference ranking loss
- not calibrated to produce consistent numeric expectations

Value models are trained with **regression to returns**, which matches the RL objective.

---

### Intuition

Think of it like writing an essay.

Reward model:

"How good is this finished essay?"

Value model:

"Given what is written so far, how good do I expect the final essay to be?"

These are different predictions.

---

### Key takeaway

Even though the **same final reward supervises the value model**, the value function learns the **expected reward of partial prefixes under the current policy**, which the reward model cannot represent.

# PPO (stable updates + token-level credit)

## A. PPO idea #1: do not move the policy too far (the ratio + clipping)

Define the ratio:

$$\rho_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\text{old}}(a_t | s_t)} = \exp(\log \pi_\theta - \log \pi_{\text{old}}).$$

PPO optimizes a clipped objective that prevents  $\rho_t$  from leaving  $[1 - \epsilon, 1 + \epsilon]$  in a way that would create overly large updates.

- $\pi_{\text{old}}$  is the policy that generated the batch.
- You can do multiple gradient steps on the same batch because importance ratios correct for drift.
- Clipping creates a trust region that stabilizes training.

## B. PPO idea #2: a learned value function gives token-level baselines

Value model predicts:

$$V(s_t) \approx \mathbb{E}[G_t | s_t].$$

Then advantage can be computed with TD or GAE (you can pick one to focus on):

- **TD residual:**

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$

- **GAE:**

$$A_t^{GAE} = \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots$$

- In single-final-reward RLHF, the final reward enters at the end, and the value function propagates it backward through advantages.
- PPO has two losses: **policy loss** (clipped) and **value loss** (fit  $V$ ).

# PPO with Temporal-Difference (TD) error

## Final advantage

$$A_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

This differs across tokens.

---

## Meaning

Was this specific action better or worse than expected at this timestep?

- Positive  $\rightarrow$  reinforce this token
- Negative  $\rightarrow$  suppress this token

# PPO with Generalized Advantage Estimation (GAE)

## Final advantage

$$A_t = \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots$$

where

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$

---

## Meaning

A smoothed measure of how much better this action was, accounting for both immediate and future prediction errors.

- $\lambda$  controls how far the reward propagates backward.
- Provides smoother and lower-variance credit assignment.

# Core PPO policy objective (clipped)

Define the ratio:

$$\rho_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\text{old}}(a_t | s_t)}.$$

The PPO clipped objective is:

$$L^{\text{policy}}(\theta) = \mathbb{E} \left[ \min \left( \rho_t(\theta) A_t, \text{clip}(\rho_t(\theta), 1 - \epsilon, 1 + \epsilon) A_t \right) \right]$$

This is what we maximize.

# GRPO (Group Relative Policy Optimization)

## 1. Motivation

PPO requires a **value model** to compute advantages.

Problems:

- training the value model is expensive
- value estimates can be unstable
- it doubles model size during training

GRPO removes the value model entirely.

Instead of learning a value baseline, GRPO uses **relative rewards within a group of sampled completions**.

# GRPO (cont'd)

## 2. Basic idea

For a prompt  $x$ , sample **multiple completions**:

$$y^{(1)}, y^{(2)}, \dots, y^{(K)}$$

Each completion gets a reward:

$$R^{(1)}, R^{(2)}, \dots, R^{(K)}$$

Now compute a **group-normalized advantage**.

# GRPO (cont'd)

## 3. GRPO advantage

First compute group statistics:

$$\mu = \frac{1}{K} \sum_{k=1}^K R^{(k)}$$

$$\sigma = \sqrt{\frac{1}{K} \sum_{k=1}^K (R^{(k)} - \mu)^2}$$

Then define advantage:

$$A^{(k)} = \frac{R^{(k)} - \mu}{\sigma}$$

So each completion is compared **relative to the group**.

## GRPO (cont'd)

GRPO still uses the **PPO clipped objective**, but the advantage is sequence-level.

For each token:

$$L^{GRPO} = \sum_t \min \left( \rho_t A^{(k)}, \text{clip}(\rho_t, 1 - \epsilon, 1 + \epsilon) A^{(k)} \right)$$

where

$$\rho_t = \frac{\pi_\theta(a_t | s_t)}{\pi_{\text{old}}(a_t | s_t)}$$

# Comparison of Policy Gradient Methods (with KL)

Method	Advantage definition	Token-level advantage?	Value model
REINFORCE	$A = R - b$	✗ No	✗ No
RLOO	$A^{(k)} = R^{(k)} - \frac{1}{K-1} \sum_{i \neq k} R^{(i)}$	✗ No	✗ No
PPO	$A_t$ from TD / GAE	✓ Yes	✓ Yes
GRPO	$A^{(k)} = \frac{R^{(k)} - \mu}{\sigma}$	✗ No	✗ No

# Comparison of Policy Gradient Methods (with KL)

Method	Policy objective per token $L_t$	KL usage	Key idea
REINFORCE	$L_t = \log \pi_\theta(a_t   s_t) A$	Reward shaping: $R \leftarrow R - \beta \sum_t \text{KL}_t$	Reinforce tokens of high-reward sequences
RLOO	$L_t = \log \pi_\theta(a_t   s_t) A^{(k)}$	Same reward shaping with KL	Compare each completion with others
PPO	$L_t = \min(\rho_t A_t, \text{clip}(\rho_t, 1 - \epsilon, 1 + \epsilon) A_t)$	Usually per-token reward: $r_t = -\beta \text{KL}_t$	Stable updates + token-level credit
GRPO	$L_t = \min(\rho_t A^{(k)}, \text{clip}(\rho_t, 1 - \epsilon, 1 + \epsilon) A^{(k)})$	Same per-token KL penalty	PPO-style update without value model

## GRPO (cont'd)

$$J(\theta) = \frac{1}{G} \sum_{i=1}^G \frac{1}{|a_i|} \sum_{t=1}^{|a_i|} \left( \min \left( \frac{\pi_{\theta}(a_{i,t}|s_i)}{\pi_{\theta_{\text{old}}}(a_{i,t}|s_i)} A_{i,t}, \text{clip} \left( \frac{\pi_{\theta}(a_{i,t}|s_i)}{\pi_{\theta_{\text{old}}}(a_{i,t}|s_i)}, 1 - \varepsilon, 1 + \varepsilon \right) A_{i,t} \right) - \beta \mathcal{D}_{\text{KL}}(\pi_{\theta}(\cdot|s_i) \parallel \pi_{\text{ref}}(\cdot|s_i)) \right)$$

With the advantage computation for the completion index  $i$ :

$$A_i = \frac{r_i - \text{mean}(r_1, r_2, \dots, r_G)}{\text{std}(r_1, r_2, \dots, r_G)}.$$

# Group Relative Policy Optimization (GRPO) (cont'd)

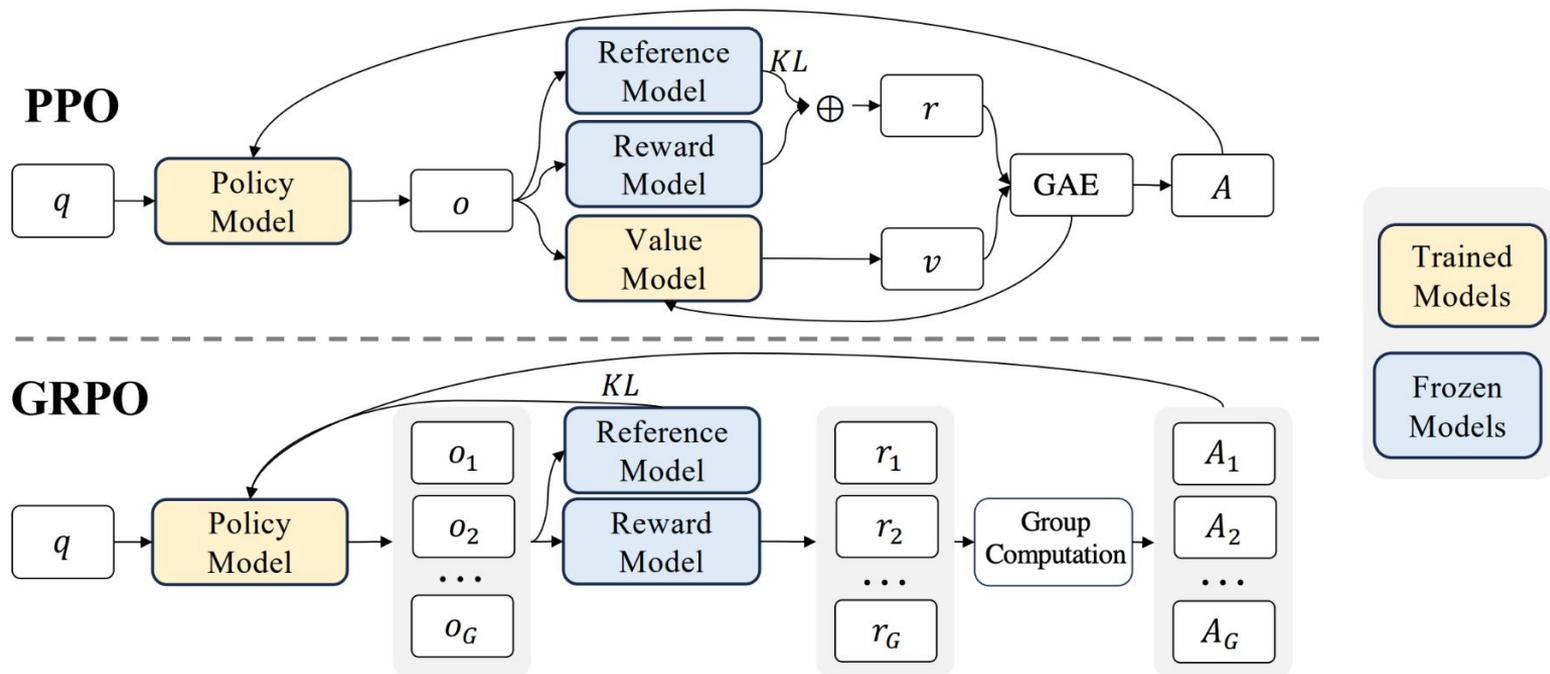


Figure 4 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.



# Large reasoning models (LRMs) / Thinking models

ChatGPT 5 Thinking ▾

GPT-5

Auto

Decides how long to think

Instant

Answers right away

Thinking ✓

Thinks longer for better answers

Pro

Research-grade intelligence

Upgrade

Legacy models >

Where should we begin?

How many NBA players have scored 60 or more points in a regular season game?

+ × Extended thinking ▾



Thinking time

Standard

Extended ✓

How many NBA players have scored 60 or more points in a regular season game?

Thinking

Skip >

THINKING,  
FAST AND SLOW

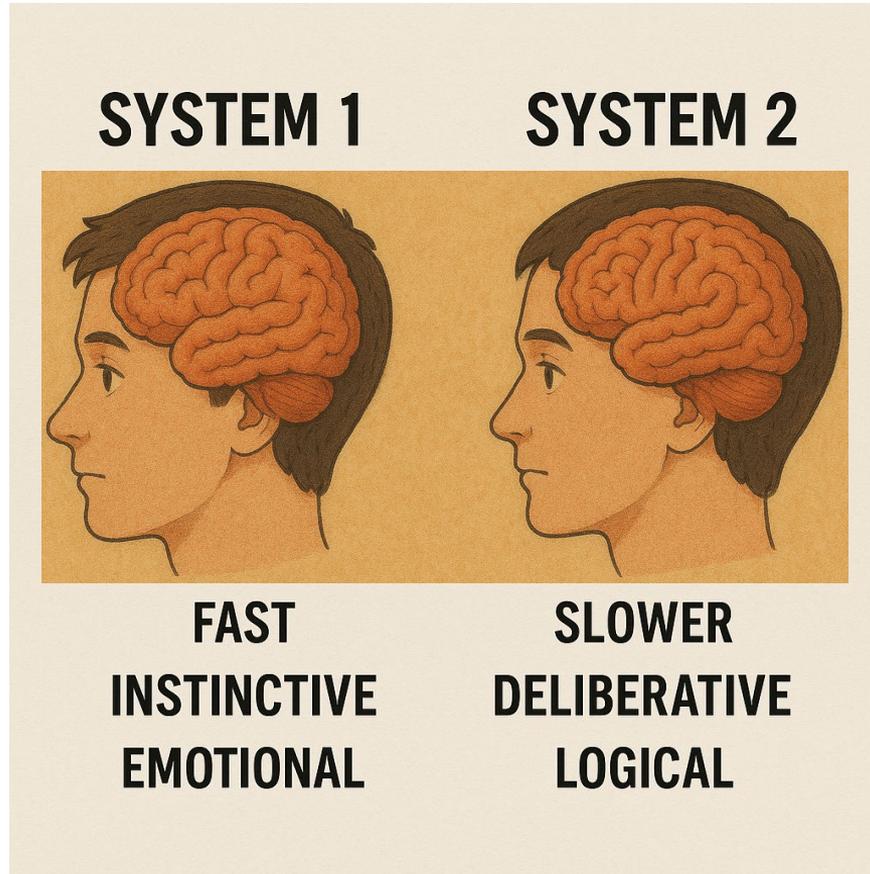


DANIEL  
KAHNEMAN

WINNER OF THE NOBEL PRIZE IN ECONOMICS

# System 1 & System 2

Which is better?



ChatGPT  
image

# OpenAI's o1

## How it works

We trained these models to spend more time thinking through problems before they respond, much like a person would. Through training, they learn to refine their thinking process, try different strategies, and recognize their mistakes.

Similar to how a human may think for a long time before responding to a difficult question, o1 uses a chain of thought when attempting to solve a problem. Through reinforcement learning, o1 learns to hone its chain of thought and refine the strategies it uses. It learns to recognize and correct its mistakes. It learns to break down tricky steps into simpler ones. It learns to try a different approach when the current one isn't working. This process dramatically improves the model's ability to reason.

# Can we infer its structure and what kinds of behaviors it rewards?

- The reward model likely provides scalar feedback that evaluates both **final answer** (correctness, usefulness, safety, clarity) and **reasoning process** (depth, coherence, reliability, safety awareness)
- Human annotators or automated evaluators review model outputs on reasoning tasks

# What the reward model likely gives high scores for

## 1. Correct reasoning and factual accuracy

Outputs that reach correct conclusions through logically consistent reasoning receive higher rewards.

- On math or logic tasks: correct answers with verifiable reasoning.
- On open-ended questions: responses that are factually accurate and well supported.

## 2. Coherent, interpretable reasoning chains

Even if the chain of thought is not visible, internal reasoning steps that are internally consistent and lead to stable answers are likely rewarded.

- The model learns to reason step by step rather than jump to conclusions.
- RL training can penalize incoherent or contradictory internal reasoning trajectories.

## 3. Efficiency and confidence calibration

The model may get higher reward when it uses an appropriate amount of “thinking” (not too short or too long) and when it expresses uncertainty realistically.

- Correct but overly long reasoning could be slightly penalized.
- Overconfident incorrect answers could get low reward.

# What the reward model likely gives high scores for (cont'd)

## 4. Safety, compliance, and harmlessness

Outputs that adhere to safety policies and avoid unsafe or biased reasoning receive higher reward.

- The o1 system card emphasizes “deliberative alignment,” meaning the model is rewarded for reasoning *about* safety before answering.
- Unsafe or policy-violating reasoning steps are penalized.

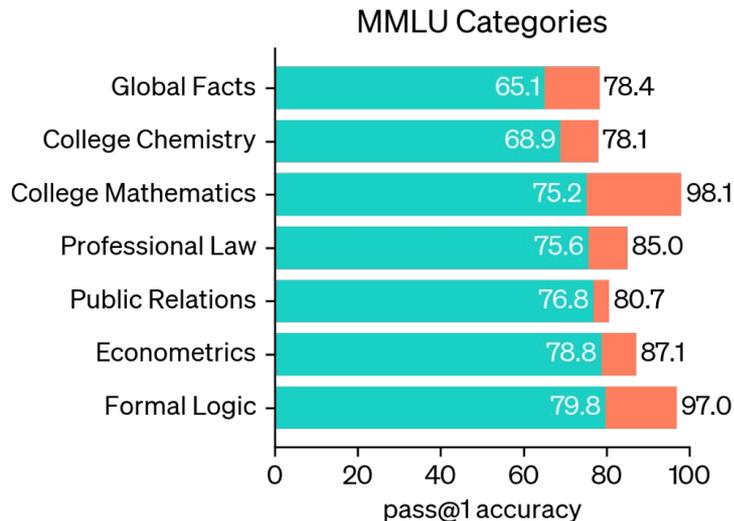
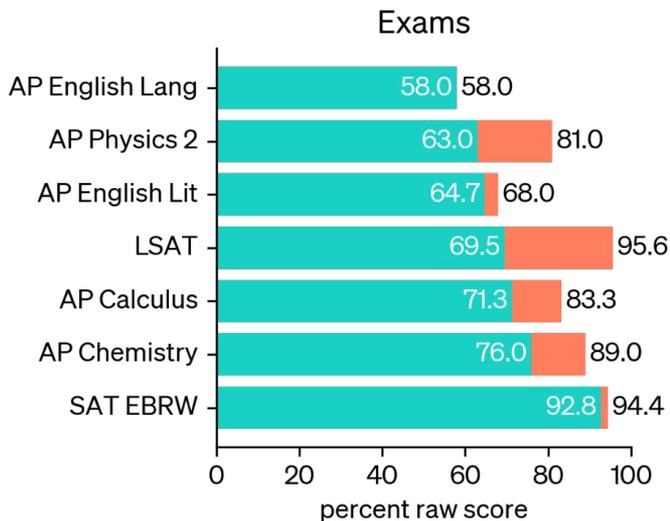
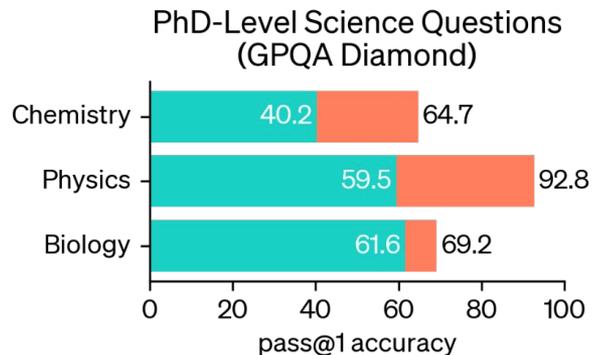
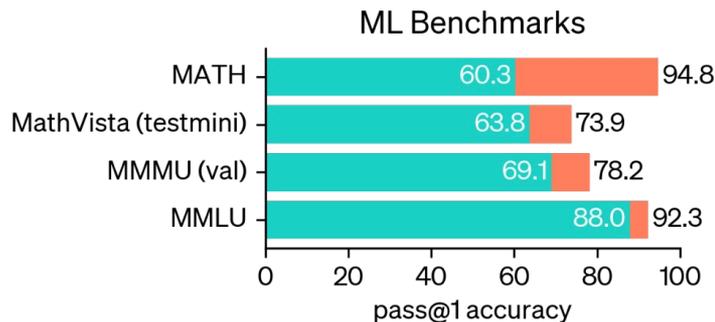
## 5. General helpfulness and clarity

The reward model also encourages clarity, helpful tone, and clear communication of reasoning results, similar to prior RLHF models.

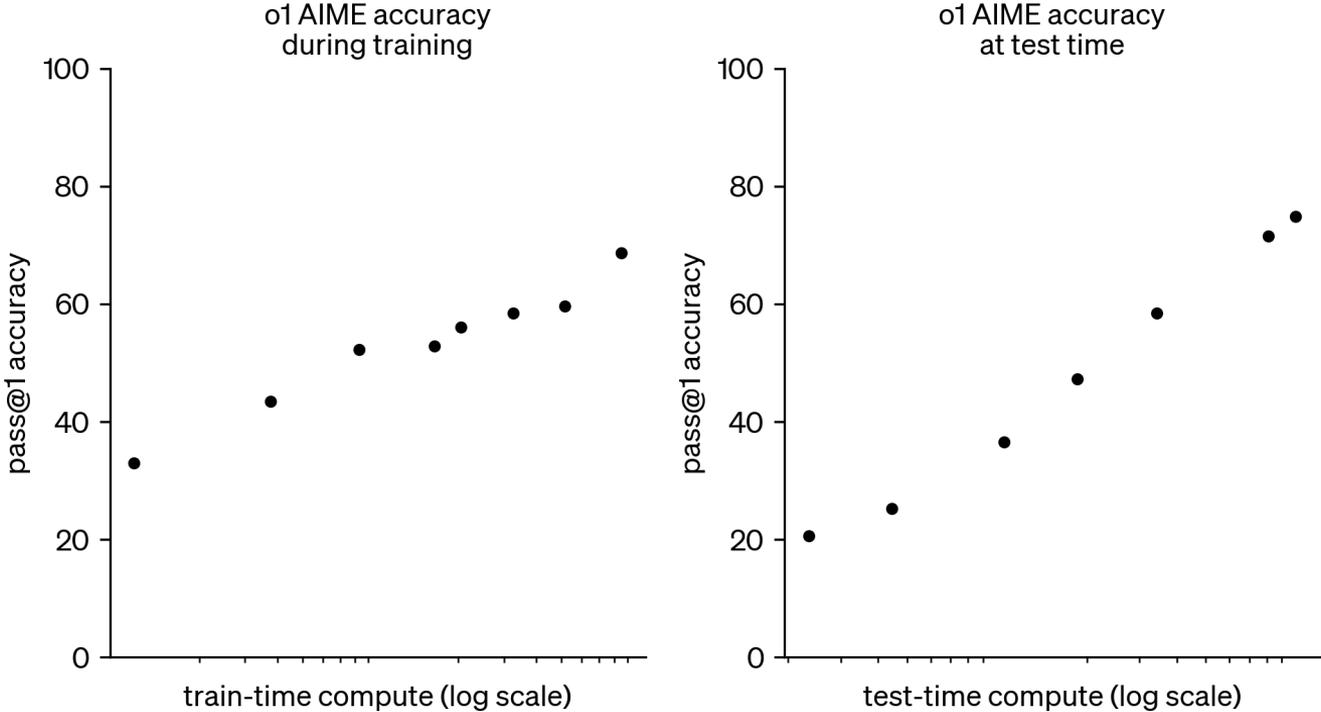
# OpenAI's o1 (cont'd)

gpt4o

o1 improvement

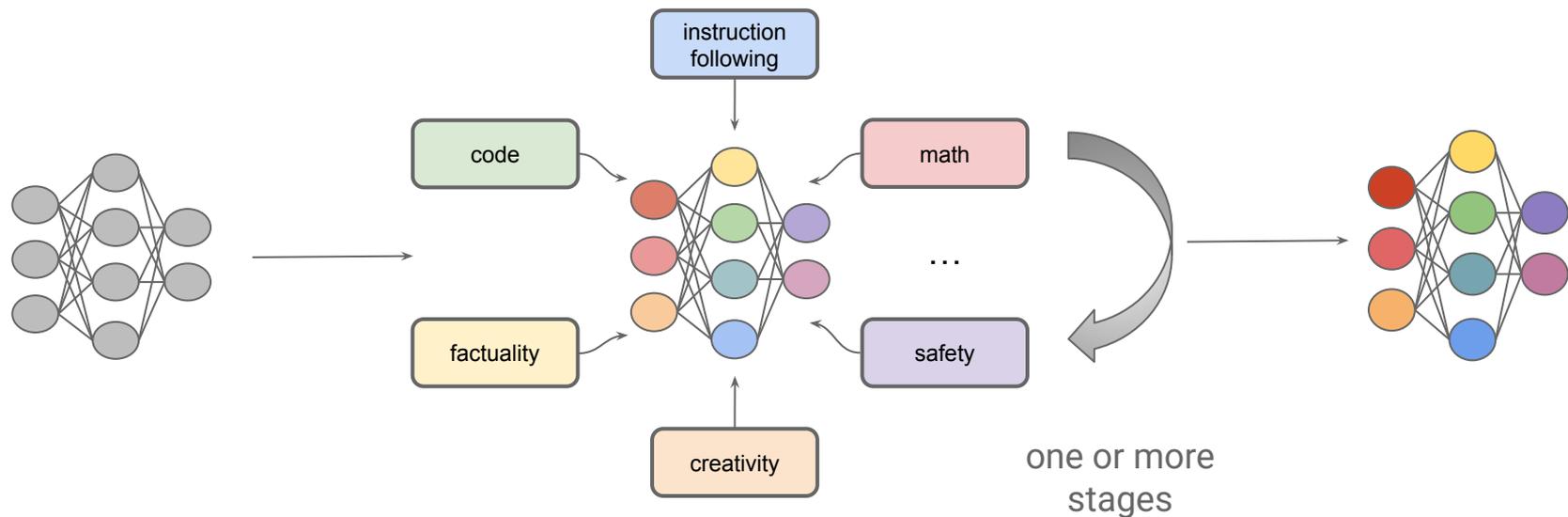


# o1 performance smoothly improves with both train-time and test-time compute



<https://openai.com/index/learning-to-reason-with-llms/>

# From System 1 to System 2



**System 1**

**Supervised Fine-tuning and/or  
Reinforcement Learning on long  
Chain-of-Thought data**

**System 2**

# DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI

`research@deepseek.com`

## Abstract

We introduce our first-generation reasoning models, DeepSeek-R1-Zero and DeepSeek-R1. DeepSeek-R1-Zero, a model trained via large-scale reinforcement learning (RL) without supervised fine-tuning (SFT) as a preliminary step, demonstrates remarkable reasoning capabilities. Through RL, DeepSeek-R1-Zero naturally emerges with numerous powerful and intriguing reasoning behaviors. However, it encounters challenges such as poor readability, and language mixing. To address these issues and further enhance reasoning performance, we introduce DeepSeek-R1, which incorporates multi-stage training and cold-start data before RL. DeepSeek-R1 achieves performance comparable to OpenAI-o1-1217 on reasoning tasks. To support the research community, we open-source DeepSeek-R1-Zero, DeepSeek-R1, and six dense models (1.5B, 7B, 8B, 14B, 32B, 70B) distilled from DeepSeek-R1 based on Owen and Llama

## DeepSeek-R1's motivation

- develop reasoning capabilities **without any supervised data (labeled input-output pairs)**
- focus on their **self-evolution** through **a pure reinforcement learning process**

# Reinforcement Learning Algorithm

- **Group Relative Policy Optimization** (GRPO)
  - Sample outputs from the current (old) policy and then optimize the policy model by maximizing the scalar reward given by a reward model

# Reinforcement Learning from Verifiable Rewards (RLVR)

The reward is the source of the training signal, which decides the optimization direction of RL. To train DeepSeek-R1-Zero, we adopt a rule-based reward system that mainly consists of two types of rewards:

- **Accuracy rewards:** The accuracy reward model evaluates whether the response is correct. For example, in the case of math problems with deterministic results, the model is required to provide the final answer in a specified format (e.g., within a box), enabling reliable rule-based verification of correctness. Similarly, for LeetCode problems, a compiler can be used to generate feedback based on predefined test cases.
- **Format rewards:** In addition to the accuracy reward model, we employ a format reward model that enforces the model to put its thinking process between '`<think>`' and '`</think>`' tags.

# Guided Chain-of-Thought (CoT) template

---

A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within `<think> </think>` and `<answer> </answer>` tags, respectively, i.e., `<think> reasoning process here </think>` `<answer> answer here </answer>`. User: **prompt**. Assistant:

---

# Group Relative Policy Optimization (GRPO)

**Group Relative Policy Optimization** In order to save the training costs of RL, we adopt Group Relative Policy Optimization (GRPO) (Shao et al., 2024), which foregoes the critic model that is typically the same size as the policy model, and estimates the baseline from group scores instead. Specifically, for each question  $q$ , GRPO samples a group of outputs  $\{o_1, o_2, \dots, o_G\}$  from the old policy  $\pi_{\theta_{old}}$  and then optimizes the policy model  $\pi_{\theta}$  by maximizing the following objective:

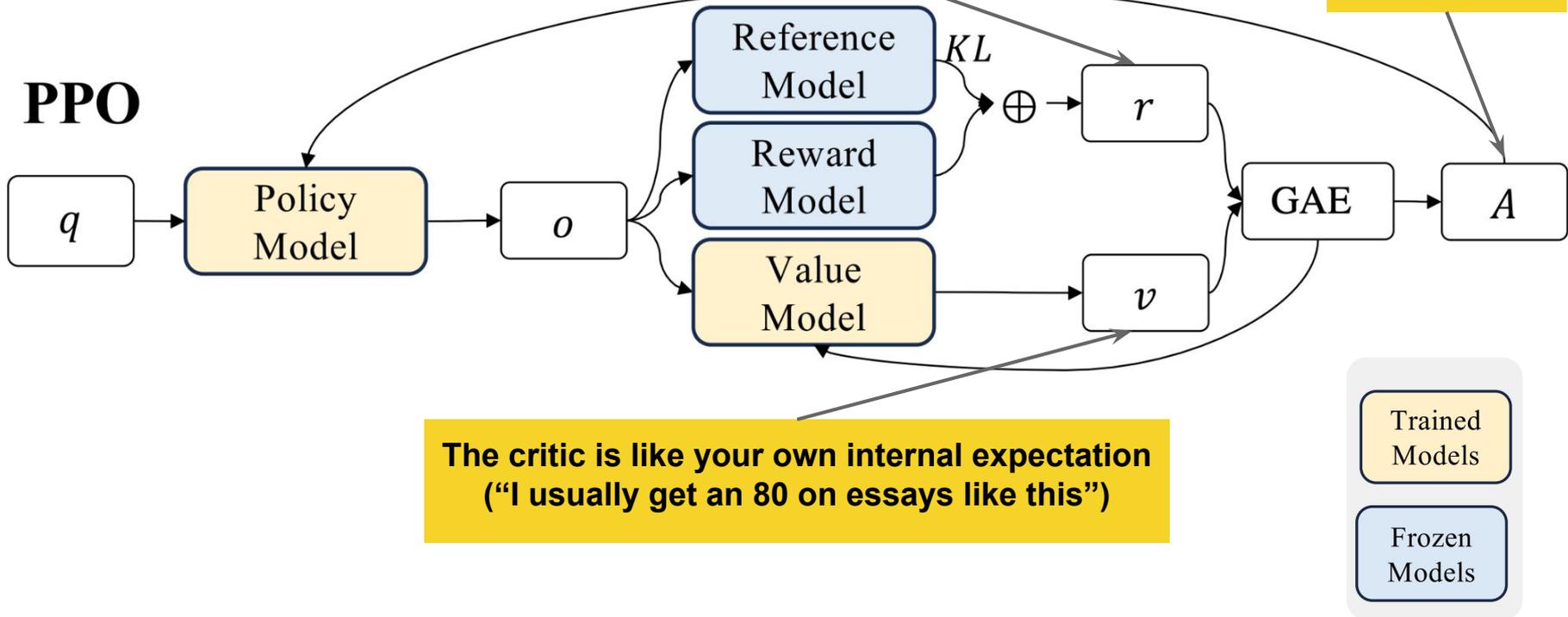
$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)] \frac{1}{G} \sum_{i=1}^G \left( \min \left( \frac{\pi_{\theta}(o_i|q)}{\pi_{\theta_{old}}(o_i|q)} A_i, \text{clip} \left( \frac{\pi_{\theta}(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1 - \varepsilon, 1 + \varepsilon \right) A_i \right) - \beta \mathbb{D}_{KL}(\pi_{\theta} || \pi_{ref}) \right), \quad (1)$$

$$\mathbb{D}_{KL}(\pi_{\theta} || \pi_{ref}) = \frac{\pi_{ref}(o_i|q)}{\pi_{\theta}(o_i|q)} - \log \frac{\pi_{ref}(o_i|q)}{\pi_{\theta}(o_i|q)} - 1, \quad (2)$$

where  $\varepsilon$  and  $\beta$  are hyper-parameters, and  $A_i$  is the advantage, computed using a group of rewards  $\{r_1, r_2, \dots, r_G\}$  corresponding to the outputs within each group:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \dots, r_G\})}{\text{std}(\{r_1, r_2, \dots, r_G\})}. \quad (3)$$

# PPO



# Group Relative Policy Optimization (GRPO) (cont'd)

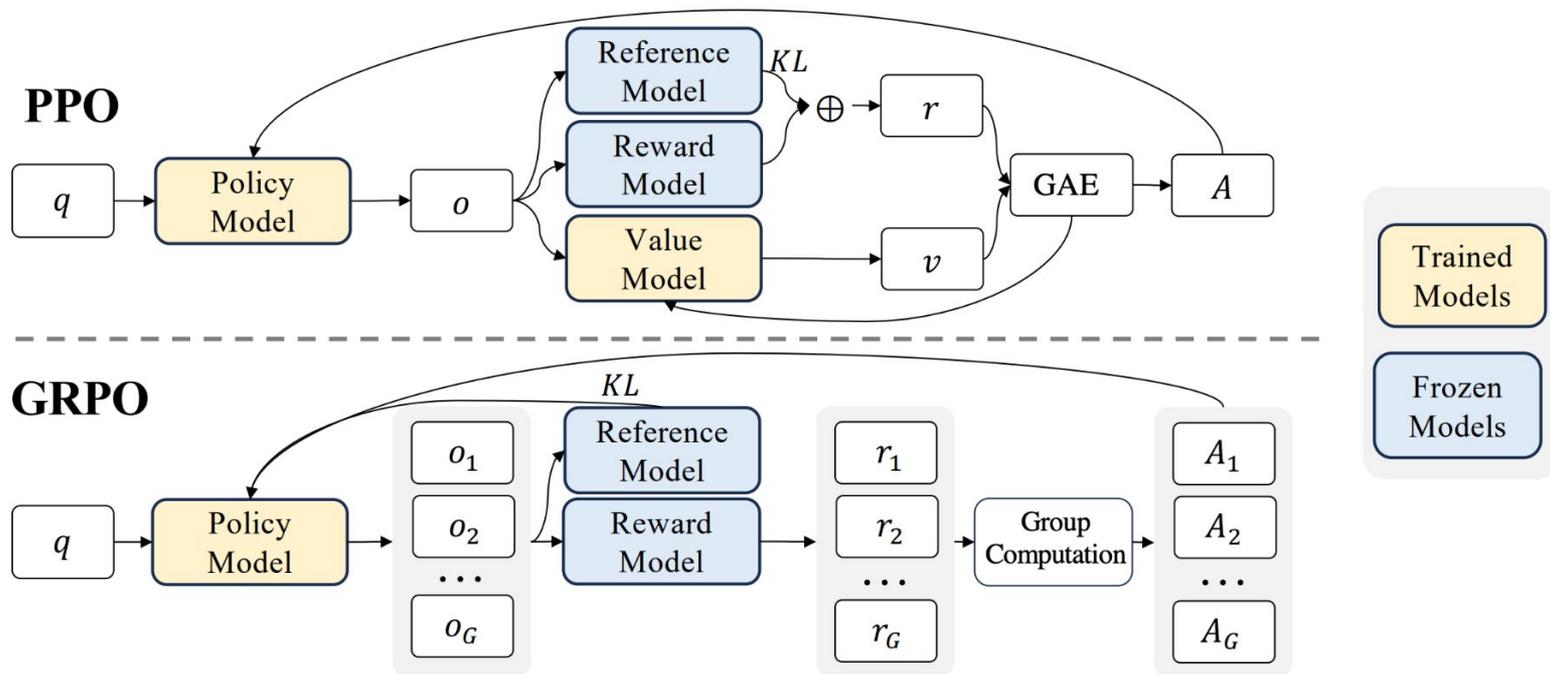


Figure 4 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

# Group advantage estimation

The relative advantage for each output  $o_i$  is defined as:

$$A_i = \frac{r_i - \text{mean}(\{r_1, \dots, r_G\})}{\text{std}(\{r_1, \dots, r_G\})}.$$

This means that an output is considered “good” if its reward is higher than the group average.

This relative normalization stabilizes updates and avoids the need for an explicit baseline network.

# Maximizing expected reward

The primary objective is to make outputs with higher rewards (from a reward model or human feedback) more likely under the updated policy.

This is achieved through the ratio term

$$\frac{\pi_{\theta}(o_i|q)}{\pi_{\theta_{\text{old}}}(o_i|q)} A_i,$$

which increases the probability of samples with positive advantages  $A_i > 0$  and decreases it for those with negative advantages  $A_i < 0$ .

Thus, GRPO pushes the model to favor responses that receive relatively higher scores within each group.

# Maintaining policy stability

The **clipping function**

$$\text{clip} \left( \frac{\pi_{\theta}(o_i|q)}{\pi_{\theta_{\text{old}}}(o_i|q)}, 1 - \epsilon, 1 + \epsilon \right)$$

prevents large updates that could destabilize the model.

This is inherited from PPO: it ensures that the new policy does not deviate too aggressively from the previous one in a single training step.

# Controlling divergence from the reference model

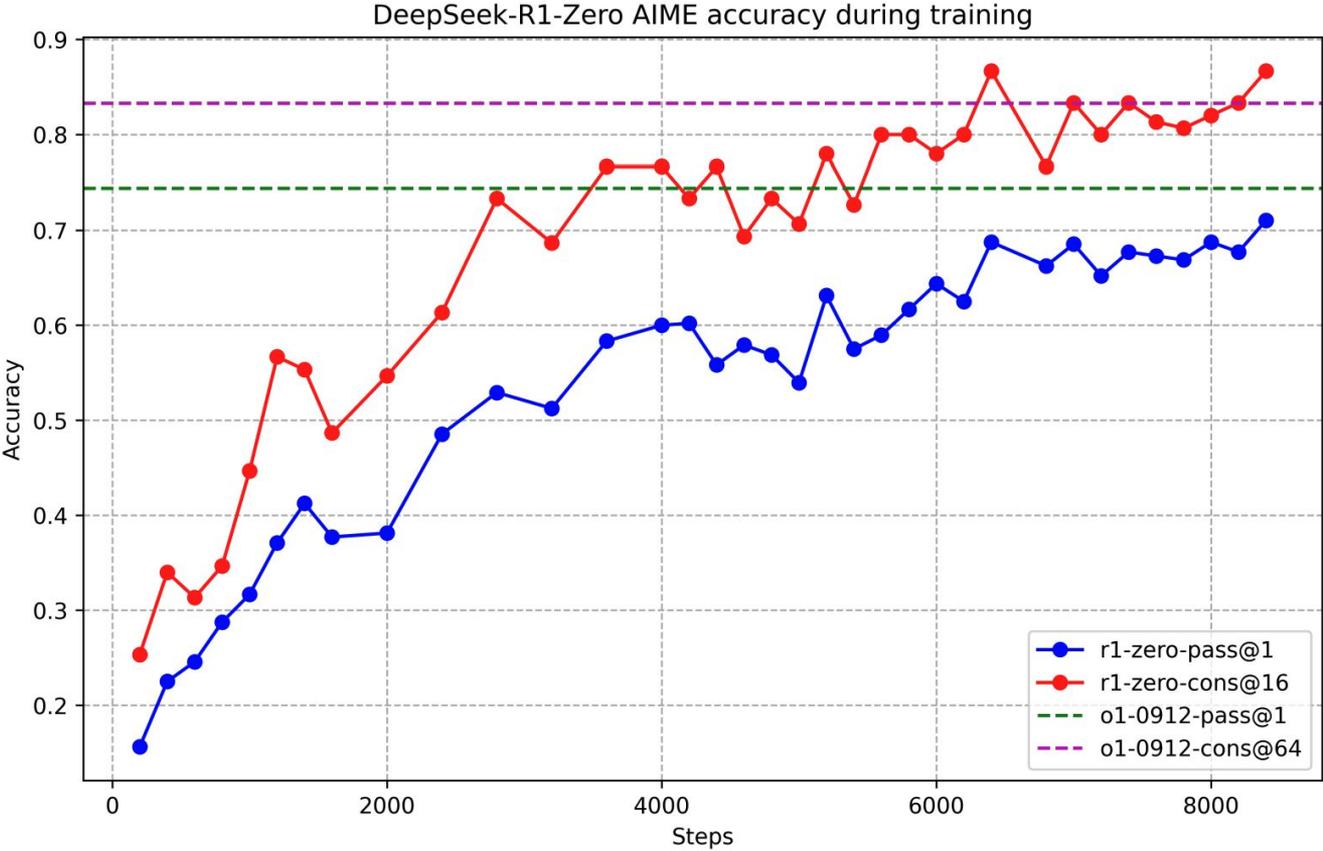
The **KL penalty term**

$$\beta D_{\text{KL}}(\pi_{\theta} \parallel \pi_{\text{ref}})$$

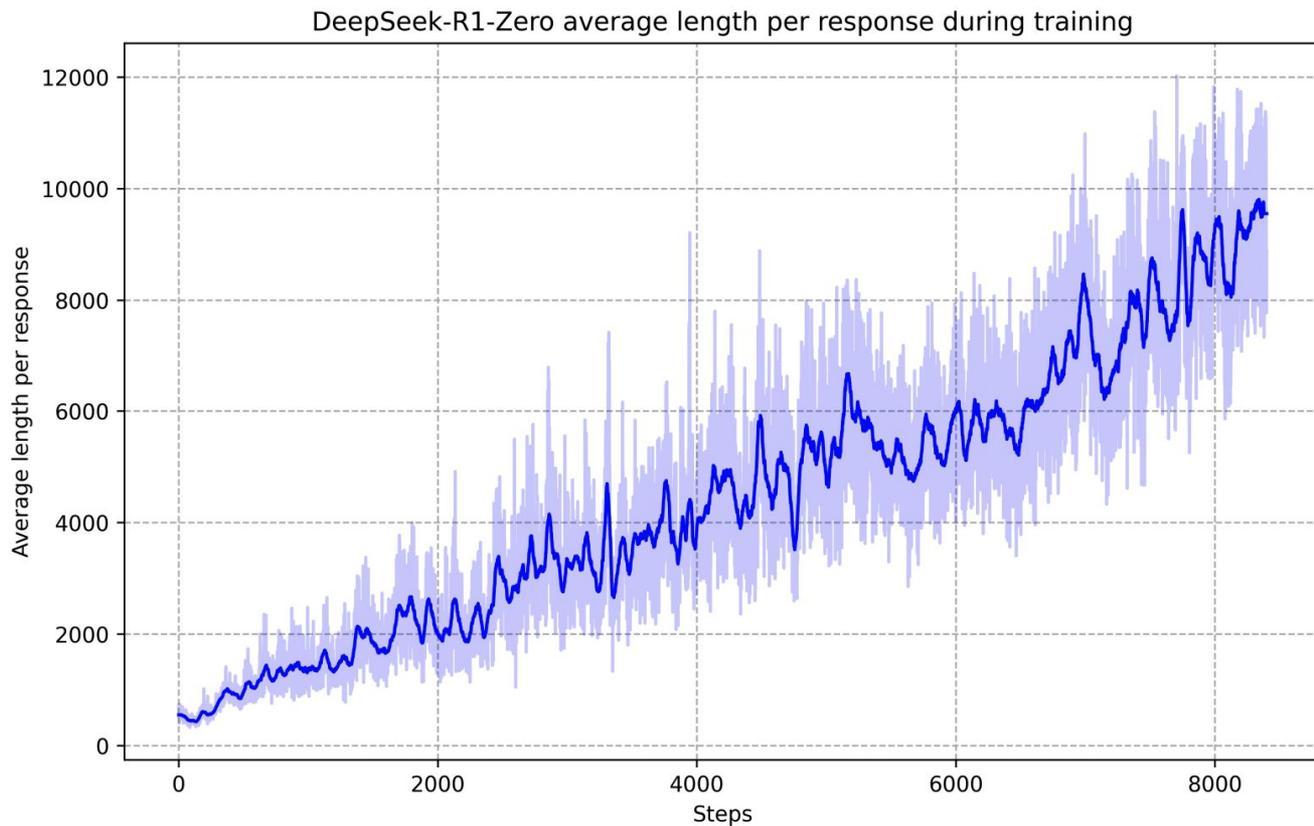
discourages the policy from diverging too far from a stable reference (often the supervised fine-tuned model).

This constraint keeps the learned behavior aligned with the base model's general language ability and prevents reward hacking or degeneration.

# Accuracy of DeepSeek-R1-Zero during training



# DeepSeek-R1-Zero naturally learns to solve reasoning tasks with more thinking time



# An interesting “aha moment” of 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  $\dots$

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

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:  $\dots$

$\dots$

---

# Problems with DeepSeek-R1-Zero

- It encounters challenges such as poor readability, and language mixing

# Problems with DeepSeek-R1-Zero

- It encounters challenges such as poor readability, and language mixing

# DeepSeek-R1

- Includes a small amount of cold-start data (thousands of long Chain-of-Thought (CoT) examples)



Imarena.ai (formerly lmsys.org)

@lmarena\_ai



Breaking News: DeepSeek-R1 surges to the top-3 in Arena 🐳!

Now ranked #3 Overall, matching the top reasoning model, o1, while being 20x cheaper and open-weight!

Highlights:

- #1 in technical domains: Hard Prompts, Coding, Math
- Joint #1 under Style Control
- MIT-licensed

A massive congrats to @deepseek\_ai for this incredible milestone and gift to the community! More analysis below 🙌

Category: Overall

Apply filter:  Style Control  Show Deprecated

Overall Questions: #models: 195 (100%) #votes: 2,572,591 (100%)

### DeepSeek-R1 #3 in Arena Overall!

Rank* (UB)	Rank (StyleCtrl)	Model	Arena Score	95% CI	Votes	Organization
1	1	Gemini-Exp-1206	1374	+5/-4	22116	Google
1	3	Gemini-2.0-Flash-Thinking-Exp-01-21	1382	+8/-6	6437	Google
3	1	ChatGPT-4o-latest_(2024-11-20)	1365	+4/-4	35328	OpenAI
3	1	DeepSeek-R1	1357	+12/-13	1883	DeepSeek
4	1	o1-2024-12-17	1352	+6/-6	9230	OpenAI
4	5	Gemini-2.0-Flash-Exp	1356	+4/-4	20939	Google

# Using reasoning models

## Get started with reasoning

Call the [Responses API](#) and specify your reasoning model and reasoning effort:

Using a reasoning model in the Responses API

python ↕ 🔗

```
1 from openai import OpenAI
2
3 client = OpenAI()
4
5 prompt = """
6 Write a bash script that takes a matrix represented as a string with
7 format '[1,2],[3,4],[5,6]' and prints the transpose in the same format.
8 """
9
10 response = client.responses.create(
11     model="gpt-5",
12     reasoning={"effort": "medium"},
13     input=[
14         {
15             "role": "user",
16             "content": prompt
17         }
18     ]
19 )
20
21 print(response.output_text)
```

**For GPT-oss: the reasoning level can be set in the system prompts, e.g., “Reasoning: high”.**



## 🌟 Train for Free

Notebooks are beginner friendly. Read our [guide](#). Add dataset, click "Run All", and export your trained model to GGUF, Ollama, vLLM or Hugging Face.

Unslot supports	Free Notebooks	Performance	Memory use
<b>gpt-oss (20B)</b>	<a href="#">▶ Start for free</a>	1.5x faster	70% less
<b>Gemma 3n (4B)</b>	<a href="#">▶ Start for free</a>	1.5x faster	50% less
<b>Qwen3 (14B)</b>	<a href="#">▶ Start for free</a>	2x faster	70% less
<b>gpt-oss (20B): GRPO</b>	<a href="#">▶ Start for free</a>	2x faster	80% less
<b>Qwen2.5-VL (7B): GSPO</b>	<a href="#">▶ Start for free</a>	1.5x faster	80% less
<b>Phi-4 (14B)</b>	<a href="#">▶ Start for free</a>	2x faster	70% less
<b>Llama 3.2 Vision (11B)</b>	<a href="#">▶ Start for free</a>	2x faster	50% less
<b>Llama 3.1 (8B)</b>	<a href="#">▶ Start for free</a>	2x faster	70% less
<b>Mistral v0.3 (7B)</b>	<a href="#">▶ Start for free</a>	2.2x faster	75% less
<b>Orpheus-TTS (3B)</b>	<a href="#">▶ Start for free</a>	1.5x faster	50% less

- See all our notebooks for: [Kaggle](#), [GRPO](#), [TTS](#) & [Vision](#)
- See [all our models](#) and [all our notebooks](#)
- See detailed documentation for Unslot [here](#)

# Test-time scaling

- Uses extra test-time compute to improve performance

# Discussion: pros and cons of test-time scaling



# Test-time scaling methods

- **Parallel (repeated sampling)**

- multiple solution attempts (run independently)
- chooses the most frequent or the best response
- [Brown et al. \(2024\)](#); [Irvine et al. \(2023\)](#); [Levi \(2024\)](#)

- **Sequential**

- later computations depend on earlier ones (e.g., a long reasoning trace)
- allows it to refine each attempt based on previous outcomes
- [Muennighoff et al.\(2025\)](#); [Snell et al. \(2024\)](#); [Hou et al. \(2025\)](#); [Lee et al. \(2025\)](#)

# Large Language Monkeys: Scaling Inference Compute with Repeated Sampling

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

<sup>†</sup>Department of Computer Science, Stanford University

<sup>‡</sup>University of Oxford

<sup>§</sup>Google DeepMind

`bradley.brown@cs.ox.ac.uk`, `bjb@stanford.edu`, `ryanehrlich@cs.stanford.edu`,  
`ronald.clark@cs.ox.ac.uk`, `qvl@google.com`, `chrismre@stanford.edu`,  
`azalia@stanford.edu`

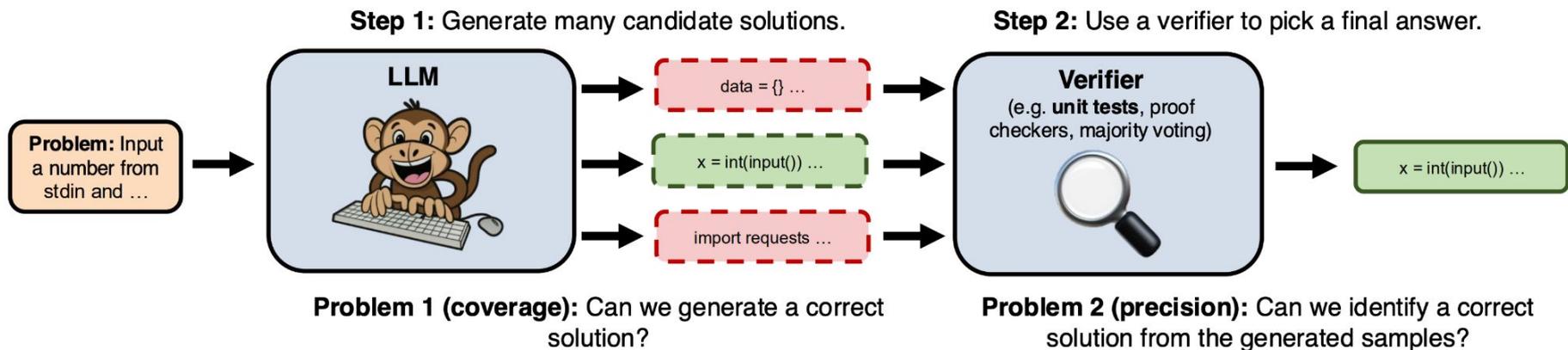


Figure 1: The repeated sampling procedure that we follow in this paper. 1) We generate many independent candidate solutions for a given problem by sampling from an LLM with a positive temperature. 2) We use a domain-specific verifier (ex. unit tests for code) to select a final answer from the generated samples.

# Measuring coverage

$$\text{pass}@k = \frac{1}{\# \text{ of problems}} \sum_{i=1}^{\# \text{ of problems}} \left[ 1 - \frac{\binom{N-C_i}{k}}{\binom{N}{k}} \right]$$

$$\text{pass}@k = \frac{1}{\# \text{ of problems}} \sum_{i=1}^{\# \text{ of problems}} \left[ 1 - \frac{\binom{N-C_i}{k}}{\binom{N}{k}} \right]$$

- $N$ : Total number of completions (outputs) generated per problem.
- $C_i$ : Number of **correct** completions among the  $N$  for problem  $i$ .
- $\binom{N}{k}$ : Number of ways to choose  $k$  outputs from  $N$ .
- $\binom{N-C_i}{k}$ : Number of ways to choose  $k$  outputs *all of which are incorrect* (i.e., chosen only from the  $N - C_i$  incorrect ones).
- So,  $\frac{\binom{N-C_i}{k}}{\binom{N}{k}}$  is the **probability** that you choose **only incorrect outputs** when sampling  $k$  outputs from the  $N$  total.
- Subtracting this from 1 gives the probability that **at least one** correct solution is in the top- $k$ .

# Coverage increases as we scale the number of samples

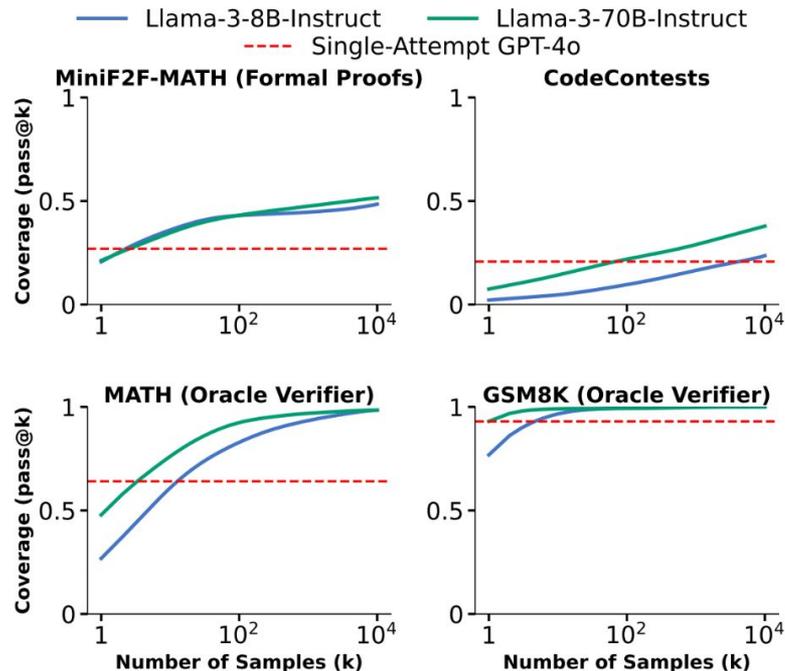
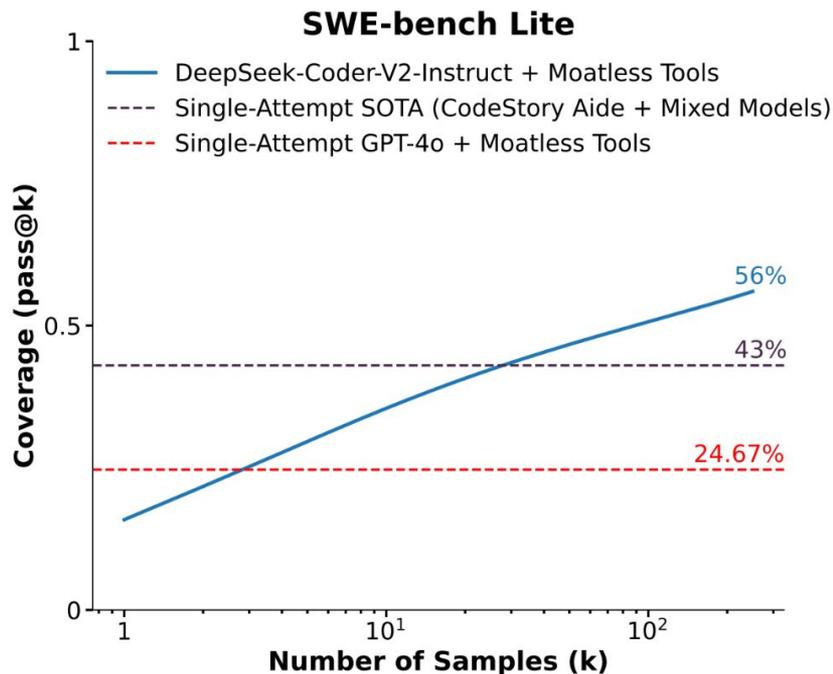


Figure 2: Across five tasks, we find that coverage (the fraction of problems solved by at least one generated sample) increases as we scale the number of samples. Notably, using repeated sampling, we are able to increase the solve rate of an open-source method from 15.9% to 56% on SWE-bench Lite.

# Scaling inference time compute via repeated sampling leads to consistent coverage gains

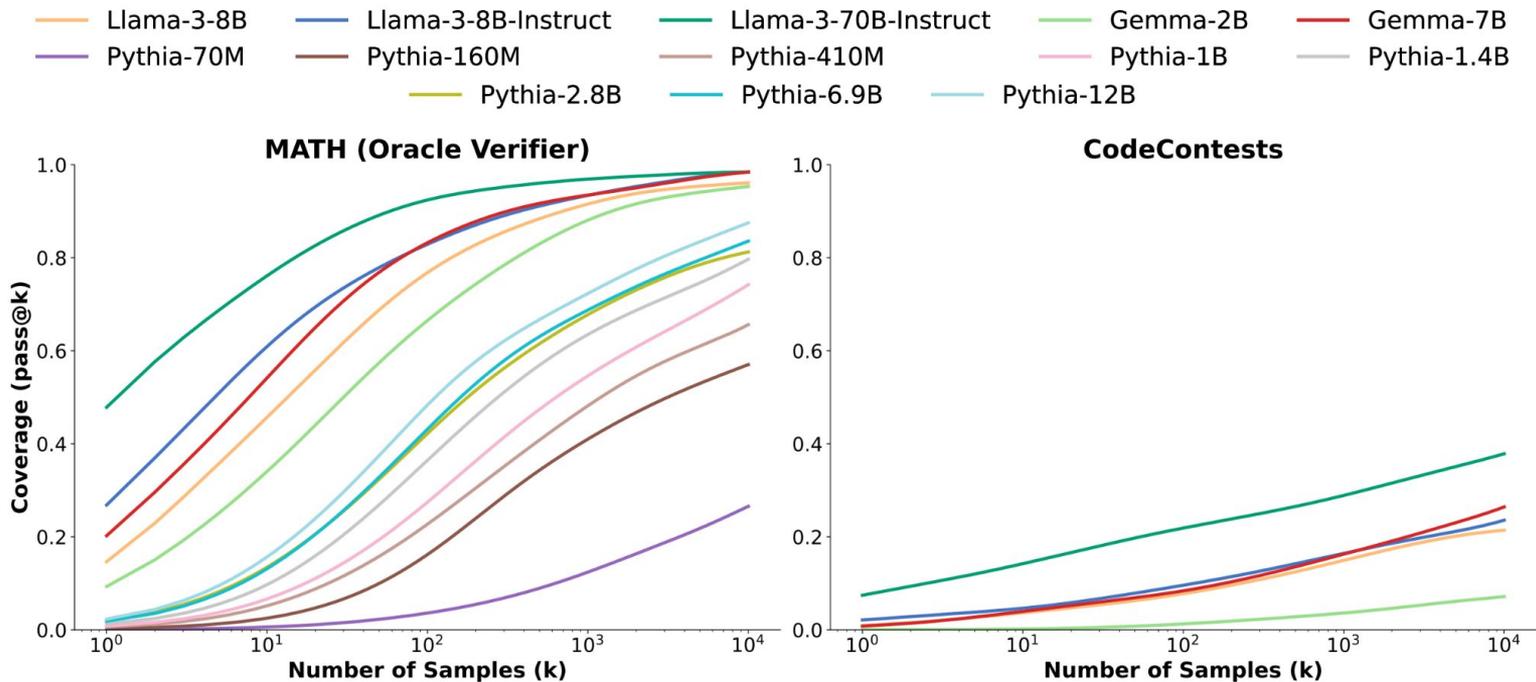


Figure 3: Scaling inference time compute via repeated sampling leads to consistent coverage gains across a variety of model sizes (70M-70B), families (Llama, Gemma and Pythia) and levels of post-training (Base and Instruct models).

# Inference FLOPs

$$\text{FLOPsPerToken}(\text{ContextLen}) \approx 2 * (\text{NumParameters} + 2 * \text{NumLayers} * \text{TokenDim} * \text{ContextLen})$$

$$\text{TotalInferenceFLOPs} \approx \left( \sum_{t=1}^{\text{NumPromptTokens}} \text{FLOPsPerToken}(t) \right) + \left( \sum_{t=1}^{\text{NumDecodeTokens}} \text{FLOPsPerToken}(t + \text{NumPromptTokens}) * \text{NumCompletions} \right)$$

# Ideal model size depends on the task, compute budget, and coverage requirements

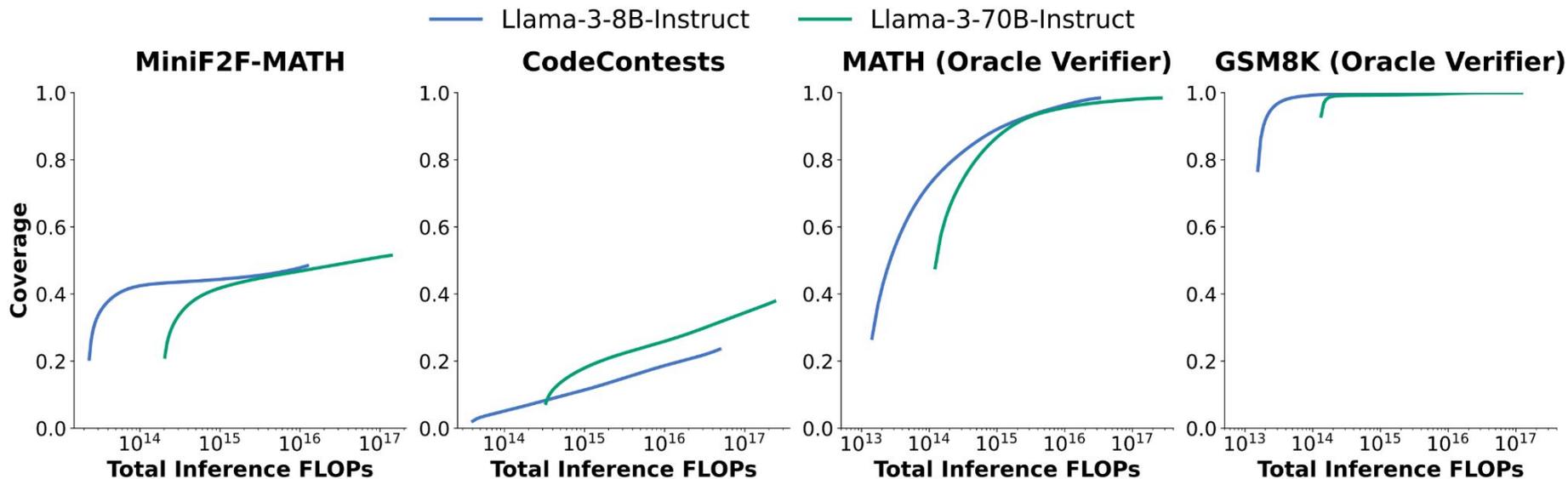


Figure 4: Comparing cost, measured in number of inference FLOPs, and coverage for Llama-3-8B-Instruct and Llama-3-70B-Instruct. We see that the ideal model size depends on the task, compute budget, and coverage requirements. Note that Llama-3-70B-Instruct does not achieve 100% coverage on GSM8K due to an incorrectly labelled ground truth answer: see Appendix E.

# API cost

Model	Cost per attempt (USD)	Number of attempts	Issues solved (%)	Total cost (USD)	Relative total cost
DeepSeek-Coder-V2-Instruct	0.0072	5	29.62	10.8	1x
GPT-4o	0.13	1	24.00	39	3.6x
Claude 3.5 Sonnet	0.17	1	26.70	51	4.7x

Table 1: Comparing API cost (in US dollars) and performance for various models on the SWE-bench Lite dataset using the Moatless Tools agent framework. When sampled more, the open-source DeepSeek-Coder-V2-Instruct model can achieve the same issue solve-rate as closed-source frontier models for under a third of the price.

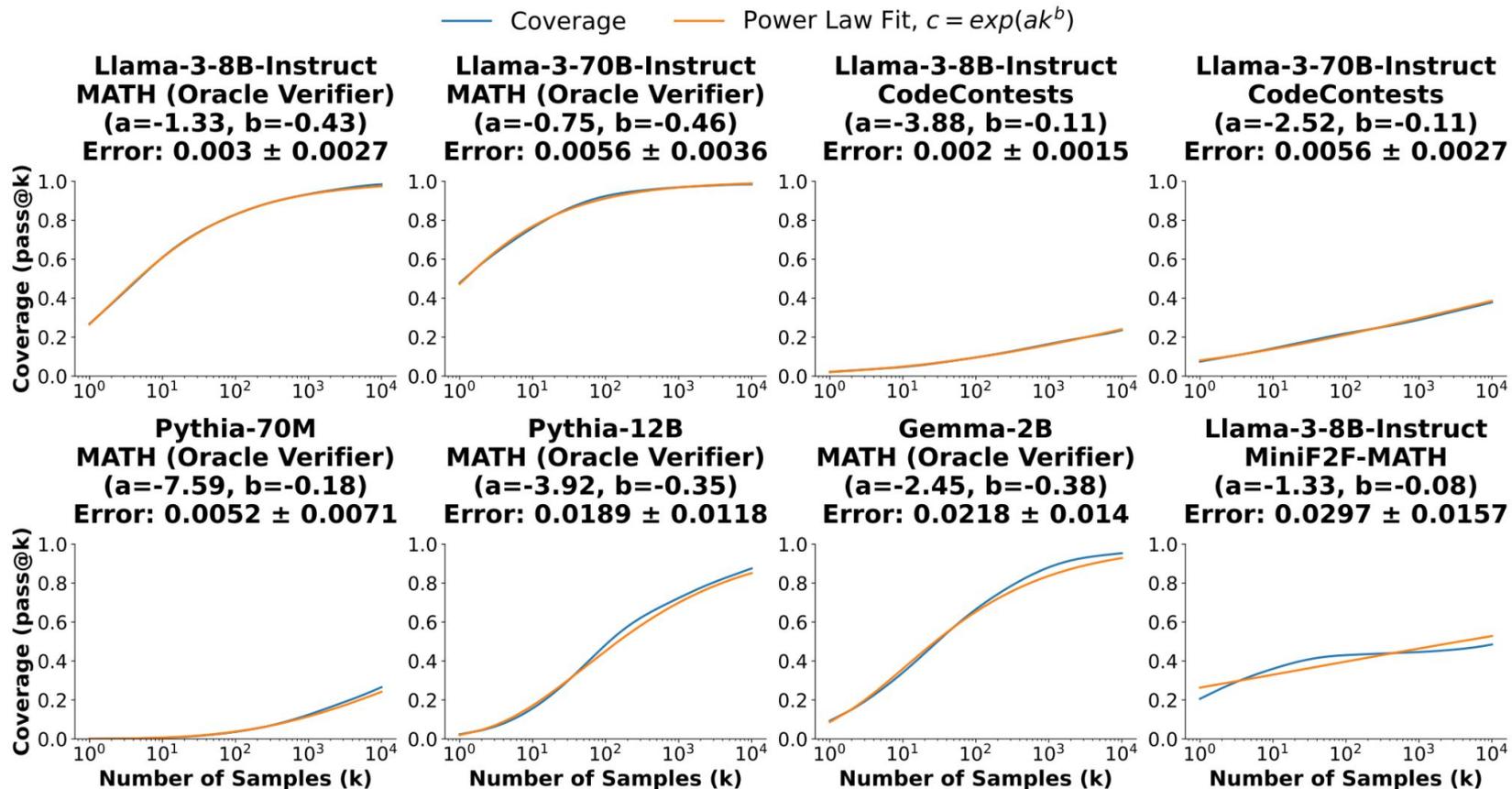


Figure 5: The relationship between coverage and the number of samples can be modelled with an exponentiated power law for most tasks and models. We highlight that some curves, such as Llama-3-8B-Instruct on MiniF2F-MATH, do not follow this trend closely. We show the mean and standard deviation of the error between the coverage curve and the power law fit across 100 evenly sampled points on the log scale.

We model the logarithm of coverage  $c$  as a function of the number of samples  $k$ :

$$\log(c) \approx ak^b$$

where  $a, b \in \mathbb{R}$  are parameters to be fitted.

To directly predict coverage  $c$ , we exponentiate both sides:

$$c \approx \exp(ak^b)$$

# The relationship between coverage and the number of samples modelled with an exponentiated power law

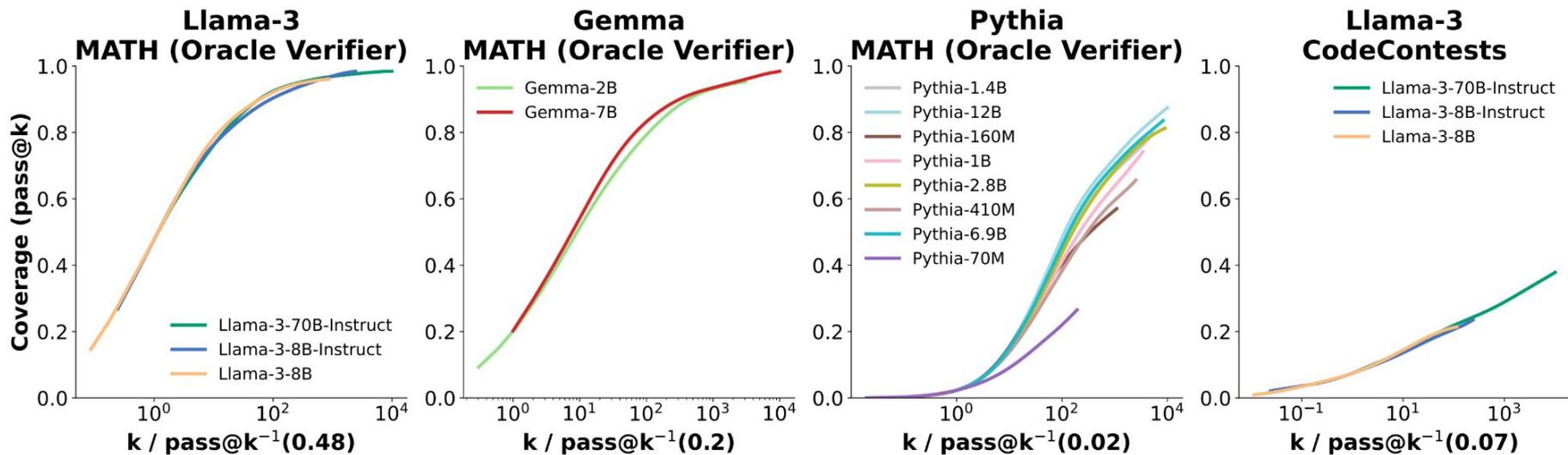


Figure 6: Overlaying the coverage curves from different models belonging to the same family. We perform this overlay by horizontally shifting every curve (with a logarithmic x-axis) so that all curves pass through the point  $(1, c)$ . We pick  $c$  to be the maximum  $\text{pass}@1$  score over all models in the plot. We note that the similarity of the curves post-shifting shows that, within a model family, sampling scaling curves follow a similar shape.

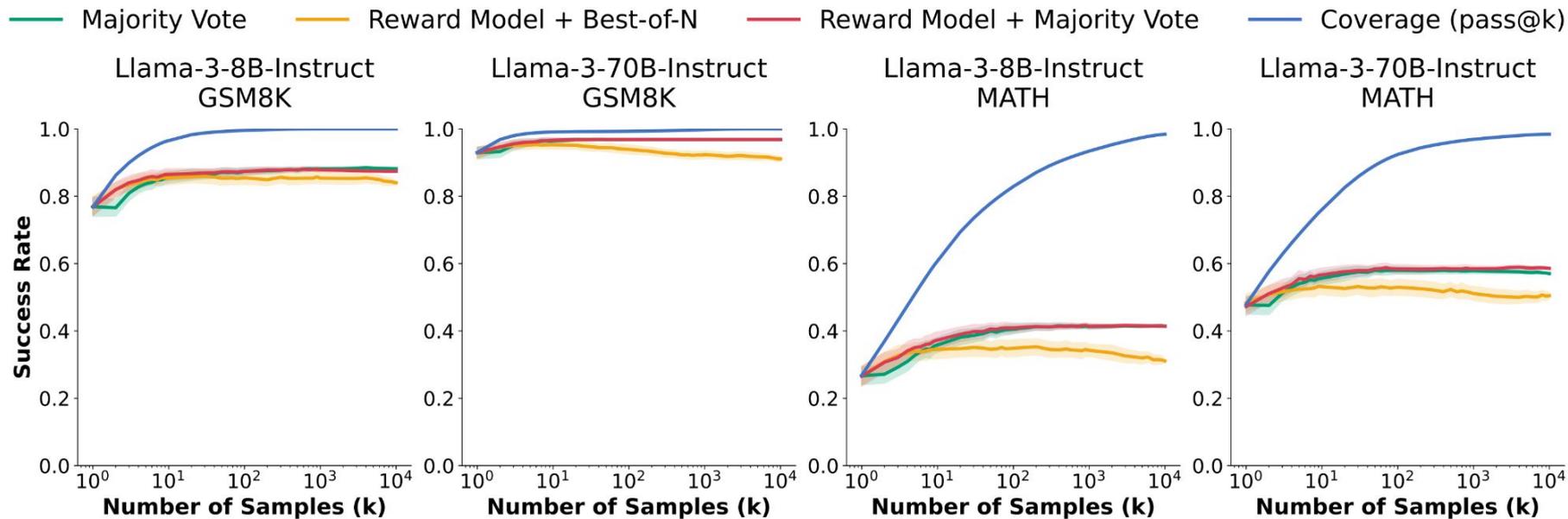


Figure 7: Comparing coverage (performance with an oracle verifier) to mainstream methods available for picking the correct answer (majority voting, reward model selection and reward model majority voting) as we increase the number of samples. Although near-perfect coverage is achieved, all sample selection methods fail to reach the coverage upper bound and saturate before reaching 100 samples. For every  $k$  value, we calculate the metric on 100 subsets of size  $k$  then plot the mean and one standard deviation across subsets.

<b>Pass@1</b>	<b># Problems</b>	<b># CoT Graded</b>	<b>Correct CoT</b>	<b>Incorrect CoT</b>	<b>Incorrect Ground Truth</b>
0-10%	5	15	11	1	1 problem, 3 CoTs
10-25%	10	30	27	3	0 problems
25-75%	29	30	28	2	0 problems
75-100%	84	30	30	0	0 problems

Table 2: Human evaluation of the validity of the Chain-of-Thought reasoning in Llama-3-8B-Instruct answers to GSM8K problems. 3 chains of thought were graded per problem. Even for difficult questions, where the model only gets  $\leq 10\%$  of samples correct, the CoTs almost always follow valid logical steps. For the model generations and human labels, see here.

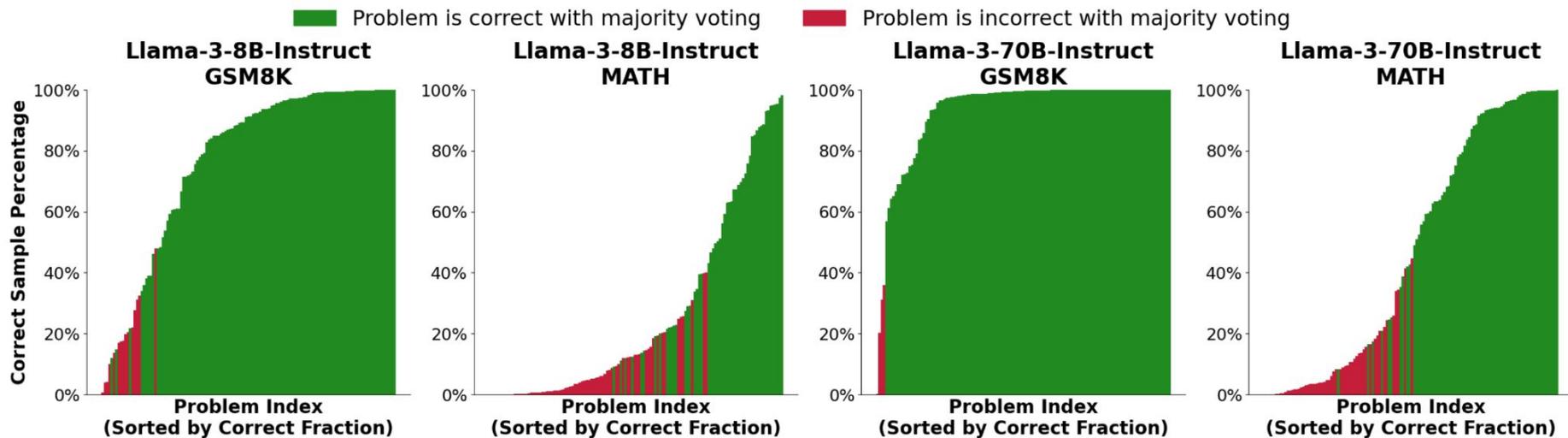


Figure 8: Bar charts showing the fraction of samples (out of 10,000 samples) that are correct, for each problem in the subsets of GSM8K and MATH we evaluate on. There is one bar per problem, and the height of the bar corresponds to the fraction of samples that arrive at the correct answer. Bars are green if self-consistency picked the correct answer and are red otherwise. We highlight that there are many problems with correct solutions, where the correct solutions are sampled infrequently.

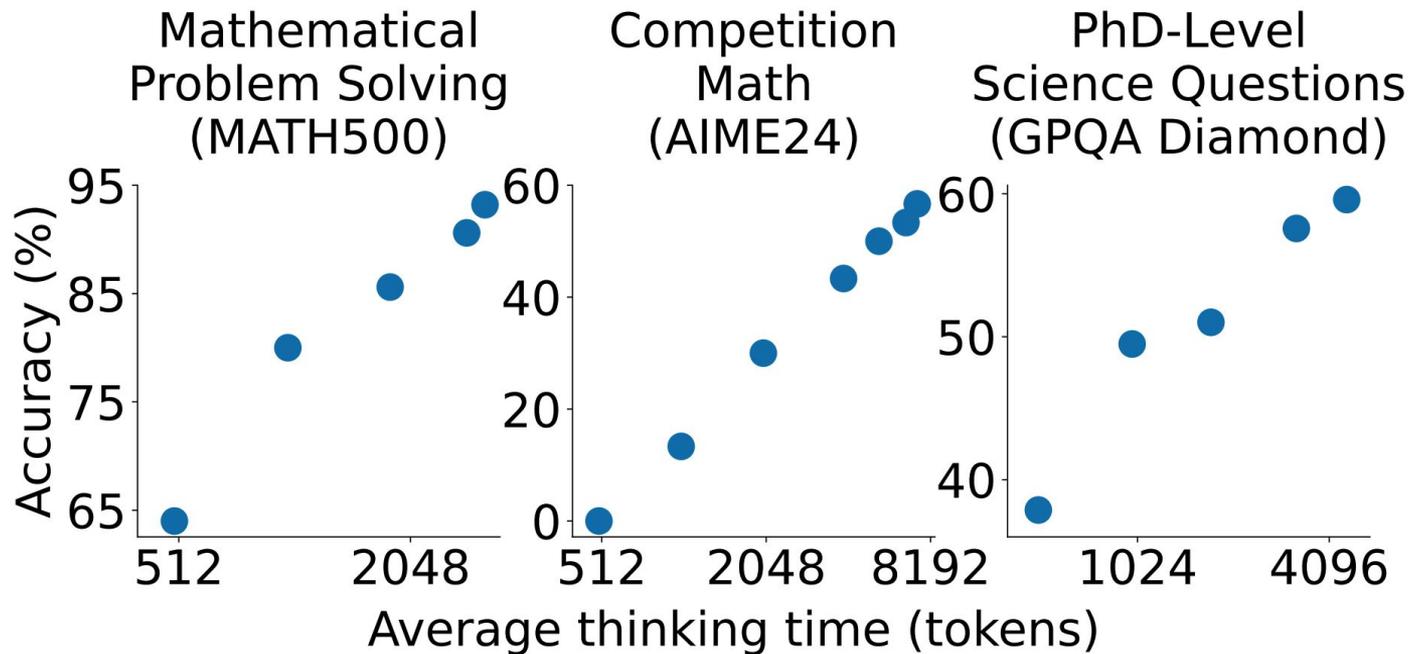
---

## **s1: Simple test-time scaling**

---

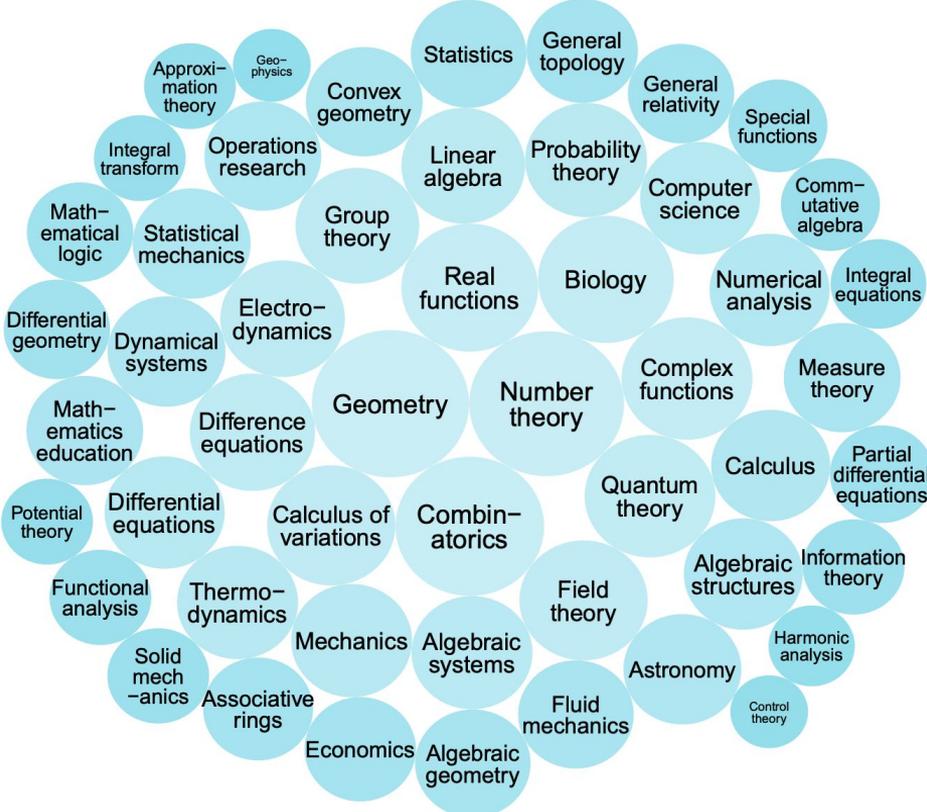
**Niklas Muennighoff<sup>\*134</sup> Zitong Yang<sup>\*1</sup> Weijia Shi<sup>\*23</sup> Xiang Lisa Li<sup>\*1</sup> Li Fei-Fei<sup>1</sup> Hannaneh Hajishirzi<sup>23</sup>  
Luke Zettlemoyer<sup>2</sup> Percy Liang<sup>1</sup> Emmanuel Candès<sup>1</sup> Tatsunori Hashimoto<sup>1</sup>**

# Test-time scaling with s1-32B

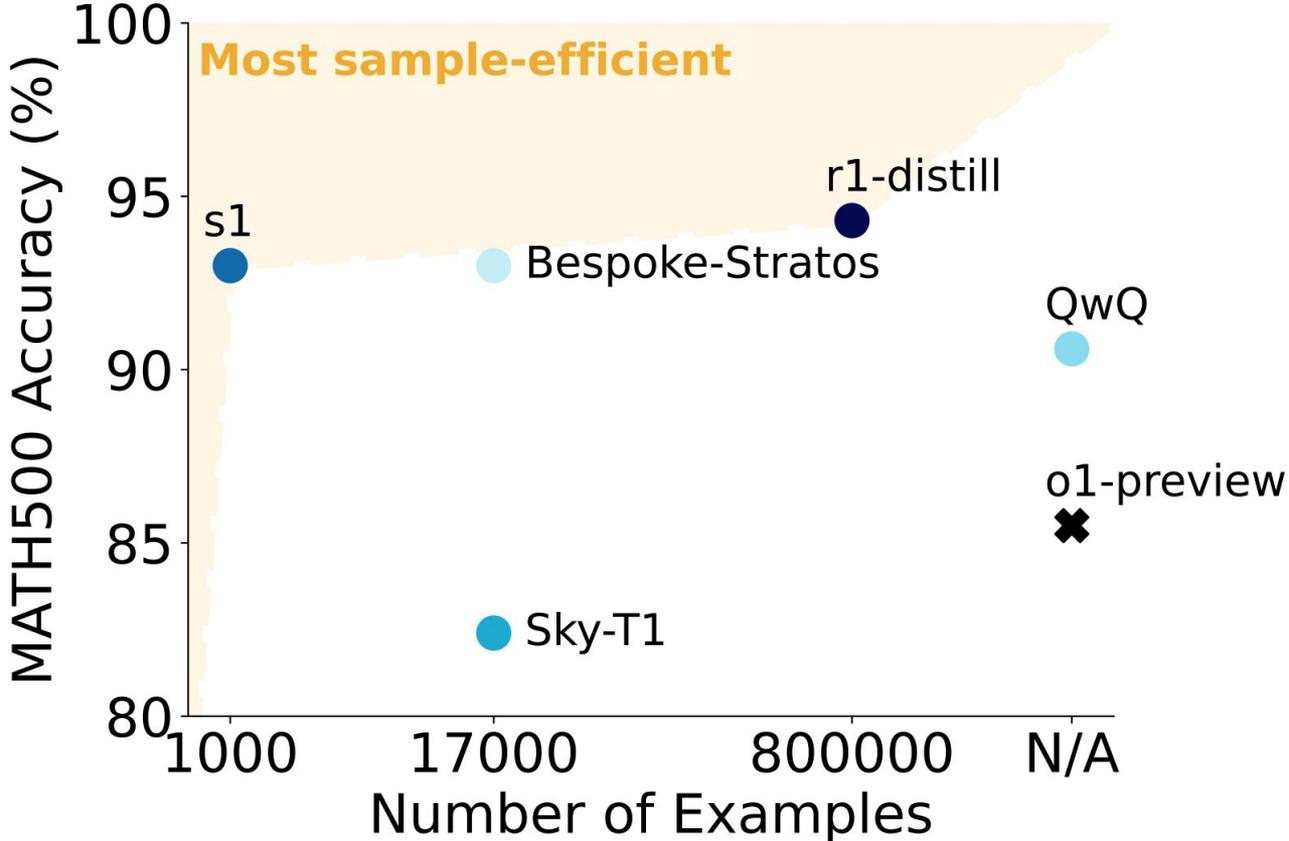


*Figure 1. Test-time scaling with s1-32B.* We benchmark s1-32B on reasoning-intensive tasks and vary test-time compute.

s1K is a dataset of 1,000 high-quality, diverse, and difficult questions with reasoning traces.



# s1-32B is on the sample-efficiency frontier



# Budget forcing

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 **3**

**Response**

For each method under consideration, we evaluate a set of tasks  $a \in \mathcal{A}$  by varying the test-time compute on a fixed benchmark (e.g., AIME24). This yields a piecewise linear function  $f$ , where the x-axis represents compute (measured in thinking tokens) and the y-axis corresponds to accuracy.

## Control

$$\text{Control} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \mathbb{I}(a_{\min} \leq a \leq a_{\max})$$

This measures the proportion of evaluations whose test-time compute (in thinking tokens) falls within a specified range  $[a_{\min}, a_{\max}]$ . In practice, we often constrain only  $a_{\max}$ . As thinking tokens reflect test-time compute, this metric quantifies how well a method supports controllability over computational cost. We report Control as a percentage, with 100% indicating perfect control.

## Scaling

$$\text{Scaling} = \frac{1}{\binom{|\mathcal{A}|}{2}} \sum_{\substack{a, b \in \mathcal{A} \\ b > a}} \frac{f(b) - f(a)}{b - a}$$

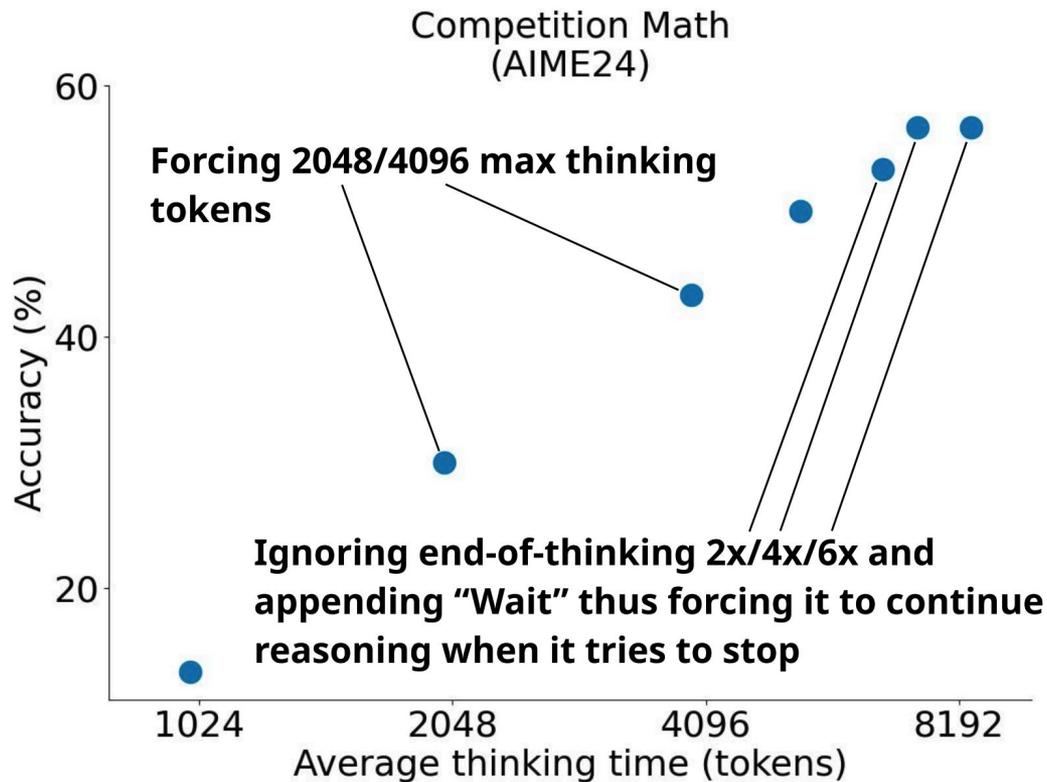
This captures the average slope of the piecewise linear function  $f$ , representing the relationship between compute and accuracy. A positive slope is necessary, and higher values indicate better scaling behavior.

## Performance

$$\text{Performance} = \max_{a \in \mathcal{A}} f(a)$$

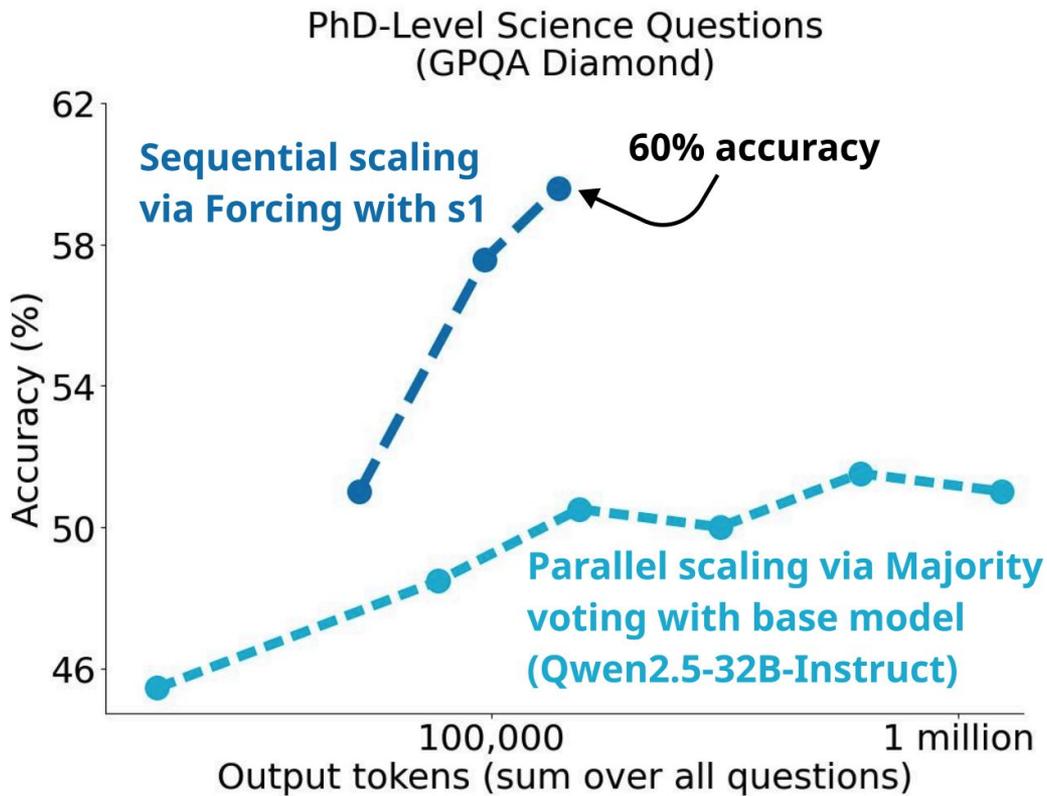
This is the highest accuracy achieved across all levels of test-time compute. In the limit, a method with consistently positive scaling will approach 100% performance.

# Budget forcing shows clear scaling trends and extrapolates to some extent



(a) Sequential scaling via budget forcing

# Parallel scaling via majority voting



(b) Parallel scaling via majority voting

# s1-32B is a strong open reasoning model

Model	# ex.	AIME 2024	MATH 500	GPQA Diamond
<b>API only</b>				
o1-preview	N.A.	44.6	85.5	73.3
o1-mini	N.A.	70.0	90.0	60.0
o1	N.A.	<b>74.4</b>	<b>94.8</b>	<b>77.3</b>
Gemini 2.0 Flash Think.	N.A.	60.0	N.A.	N.A.
<b>Open Weights</b>				
Qwen2.5- 32B-Instruct	N.A.	26.7	84.0	49.0
QwQ-32B	N.A.	50.0	90.6	54.5
r1	≫800K	<b>79.8</b>	<b>97.3</b>	<b>71.5</b>
r1-distill	800K	72.6	94.3	62.1
<b>Open Weights and Open Data</b>				
Sky-T1	17K	43.3	82.4	56.8
Bespoke-32B	17K	<b>63.3</b>	93.0	58.1
s1 w/o BF	<b>1K</b>	50.0	92.6	56.6
<b>s1-32B</b>	<b>1K</b>	56.7	<b>93.0</b>	<b>59.6</b>

# s1K data ablations

Model	AIME 2024	MATH 500	GPQA Diamond
1K-random	36.7 [-26.7%, -3.3%]	90.6 [-4.8%, 0.0%]	52.0 [-12.6%, 2.5%]
1K-diverse	26.7 [-40.0%, -10.0%]	91.2 [-4.0%, 0.2%]	54.6 [-10.1%, 5.1%]
1K-longest	33.3 [-36.7%, 0.0%]	90.4 [-5.0%, -0.2%]	59.6 [-5.1%, 10.1%]
59K-full	53.3 [-13.3%, 20.0%]	92.8 [-2.6%, 2.2%]	58.1 [-6.6%, 8.6%]
<b>s1K</b>	<b>50.0</b>	<b>93.0</b>	<b>57.6</b>

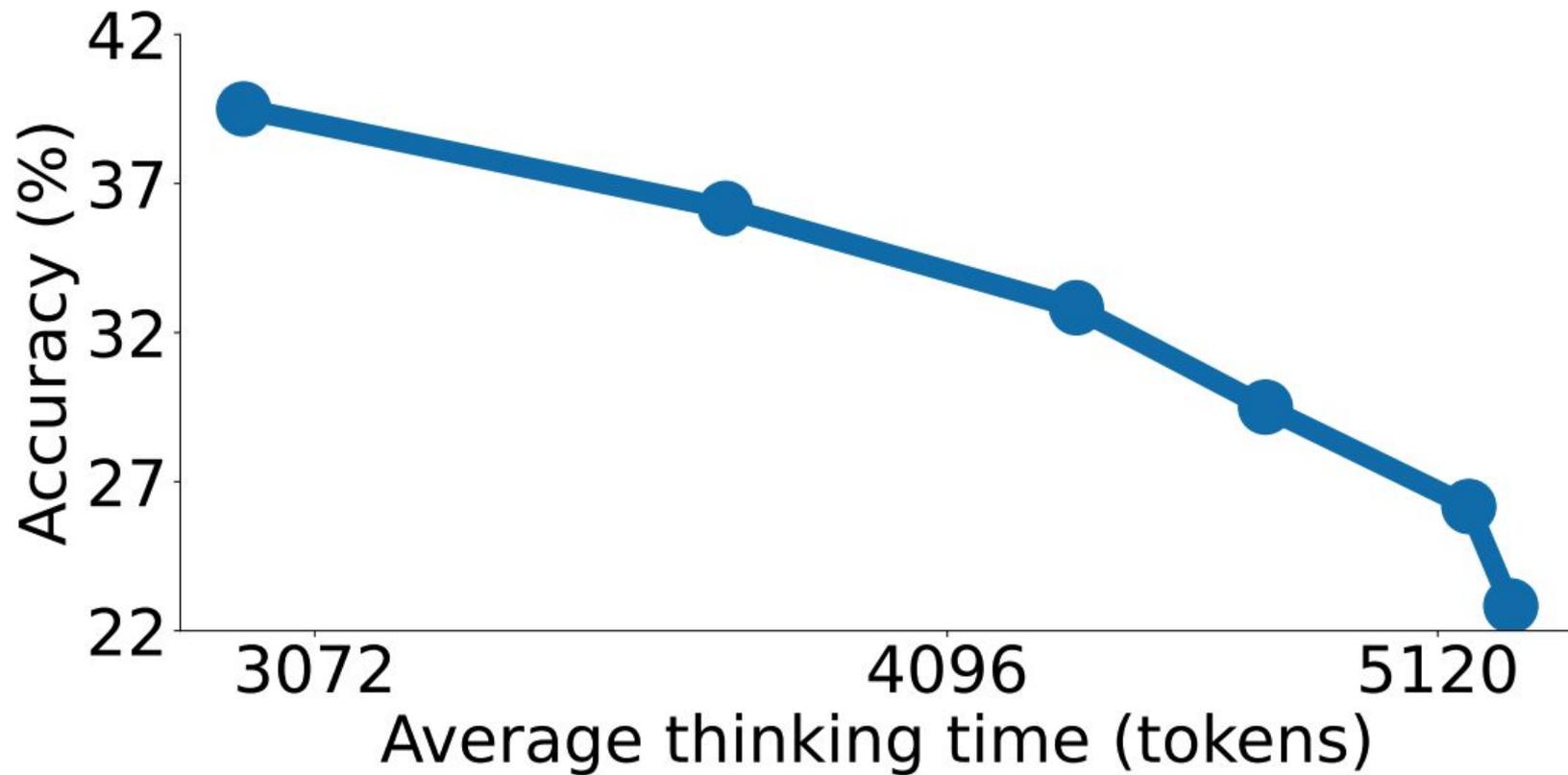
## Ablations on methods to scale test-time compute

Method	Control	Scaling	Performance	$ \mathcal{A} $
<b>BF</b>	<b>100%</b>	15	<b>56.7</b>	5
TCC	40%	-24	40.0	5
TCC + BF	<b>100%</b>	13	40.0	5
SCC	60%	3	36.7	5
SCC + BF	<b>100%</b>	6	36.7	5
CCC	50%	<b>25</b>	36.7	2
RS	<b>100%</b>	-35	40.0	5

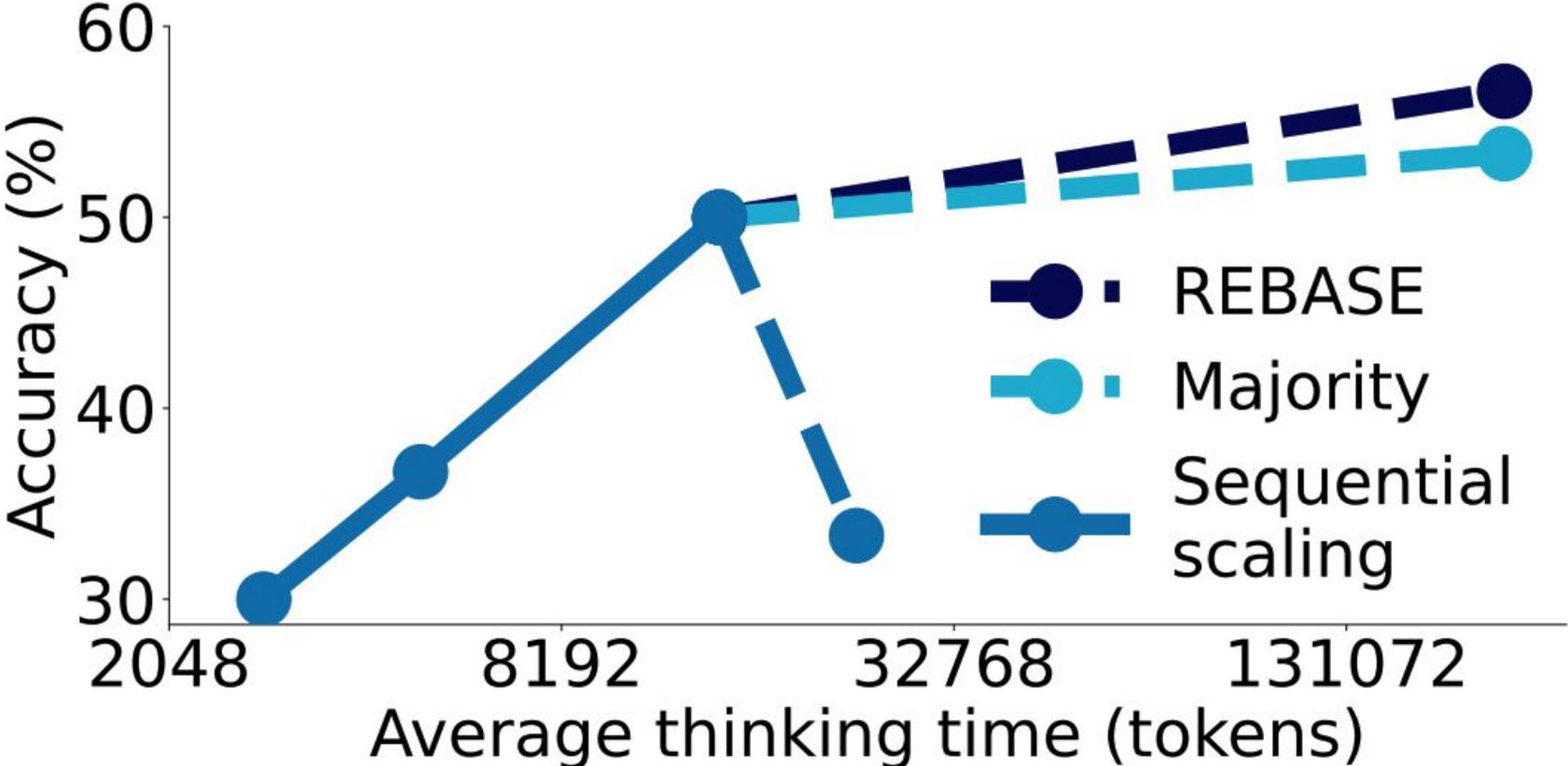
## Budget forcing extrapolation ablations

Model	AIME 2024	MATH 500	GPQA Diamond
No extrapolation	50.0	<b>93.0</b>	57.6
2x without string	50.0	90.2	55.1
2x “Alternatively”	50.0	92.2	<b>59.6</b>
2x “Hmm”	50.0	<b>93.0</b>	<b>59.6</b>
2x “Wait”	<b>53.3</b>	<b>93.0</b>	<b>59.6</b>

## Rejection sampling



# Augmenting s1 with REBASE (process reward model)



# Why does supervised fine-tuning on just 1,000 samples lead to such performance gains?

- We hypothesize that the model is already exposed to large amounts of reasoning data during pretraining which spans trillions of tokens.
- Thus, the ability to perform reasoning is already present in our model.
- Our sample-efficient fine-tuning stage just activates it and we scale it further at test time with budget forcing.

# Superficial Alignment Hypothesis

- LIMA: Less is more for alignment ([Zhou et al., 2023](#))
  - 1,000 examples can be sufficient to align a model to adhere to user preferences

**Thank you!**