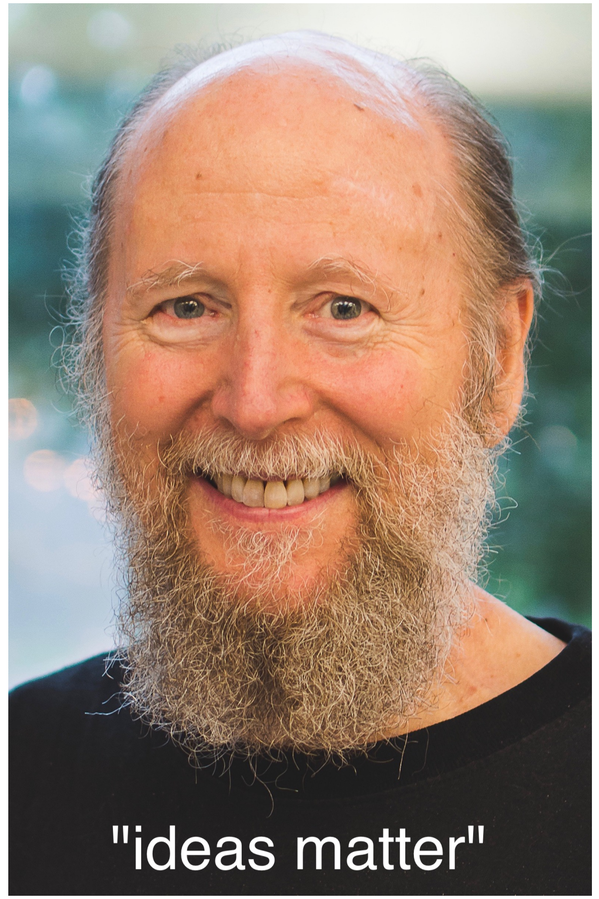
Scaling RL and Inference

27 Oct. 2025, Reasoning in AI

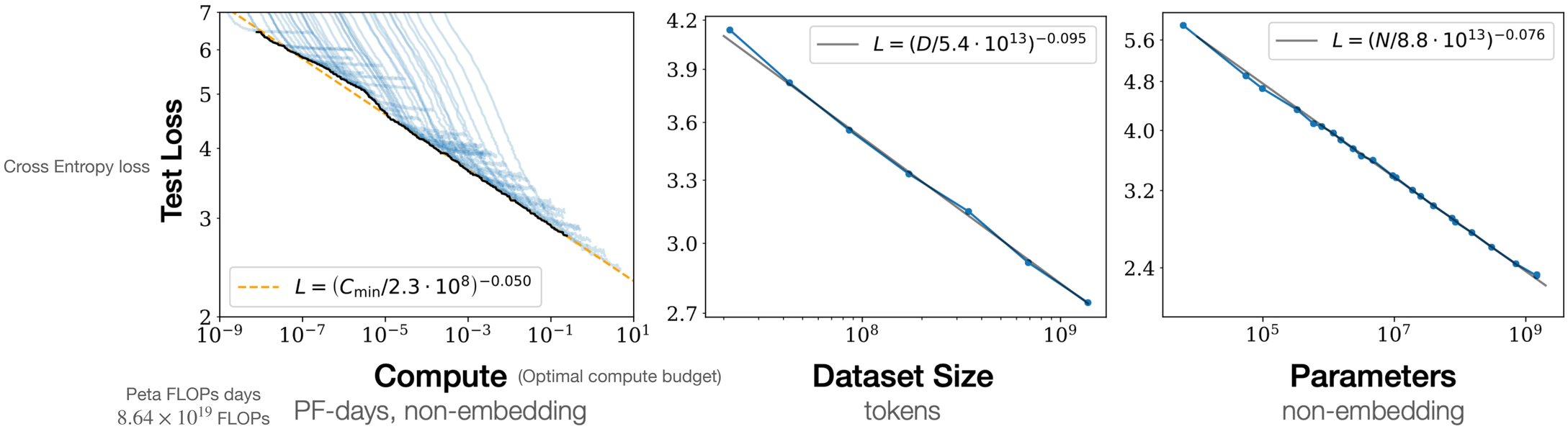
The Bitter Lesson

The bitter lesson is based on the historical observations that 1) AI researchers have often tried to build knowledge into their agents, 2) this always helps in the short term, and is personally satisfying to the researcher, but 3) in the long run it plateaus and even inhibits further progress, and 4) breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning.



Richard Sutton

Scaling training improves the performance



- "Performance has a power-law relationship with each factor when not bottlenecked by the other two."
- Issue: scaling the training phase is limited, in particular, by the data.
  - With limited data, models would memorizing the data after some time…

[Kaplan et al., 2020]

Continue to scale

- What else can we scale?
  - Scaling the training phase is limited because of the data.
  - We can scale the RL phase (as the model discovers solutions itself).
  - We can scale model's compute at inference time.

Scaling RL compute

The Art of Scaling Reinforcement Learning Compute for LLMs

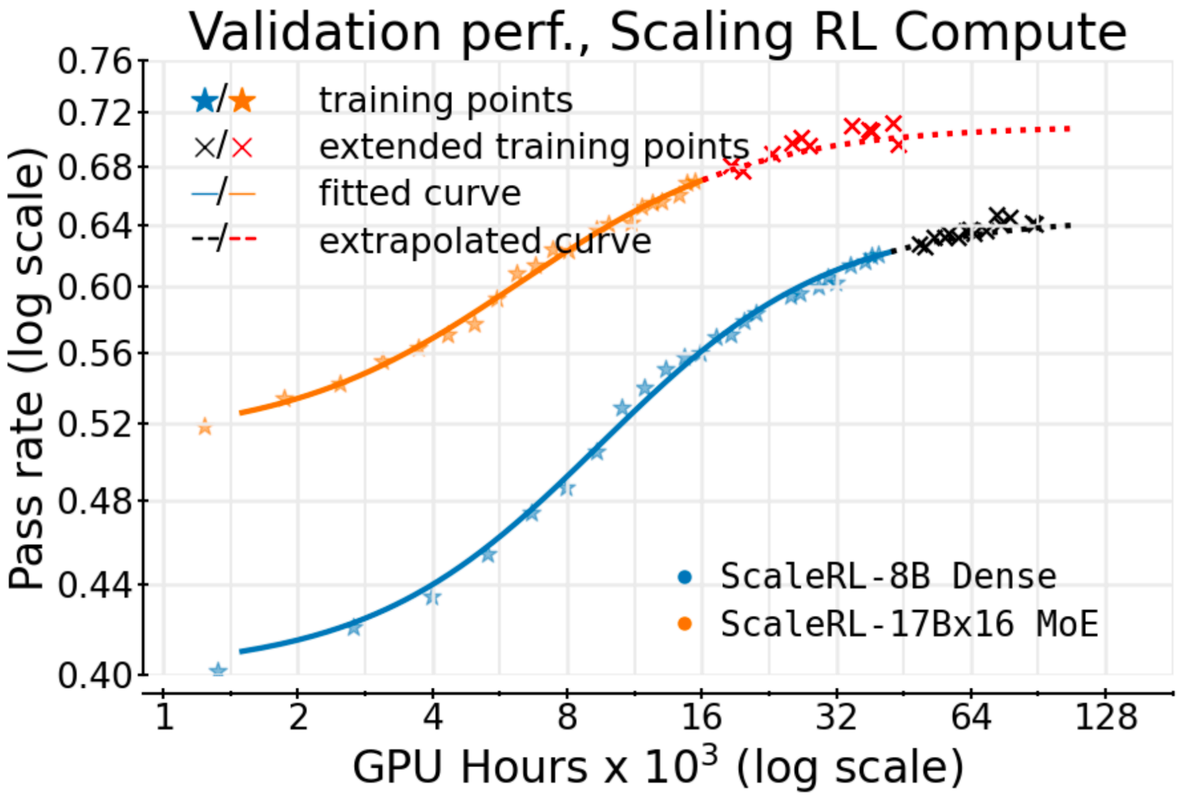
Devvrit Khatri[2,*,†], **Lovish Madaan**[1,3,*],
Rishabh Tiwari[4†], **Rachit Bansal**[5†], **Sai Surya Duvvuri**[2†], **Manzil Zaheer**[1†],
Inderjit S. Dhillon[2], **David Brandfonbrener**[1], **Rishabh Agarwal**[6,†]

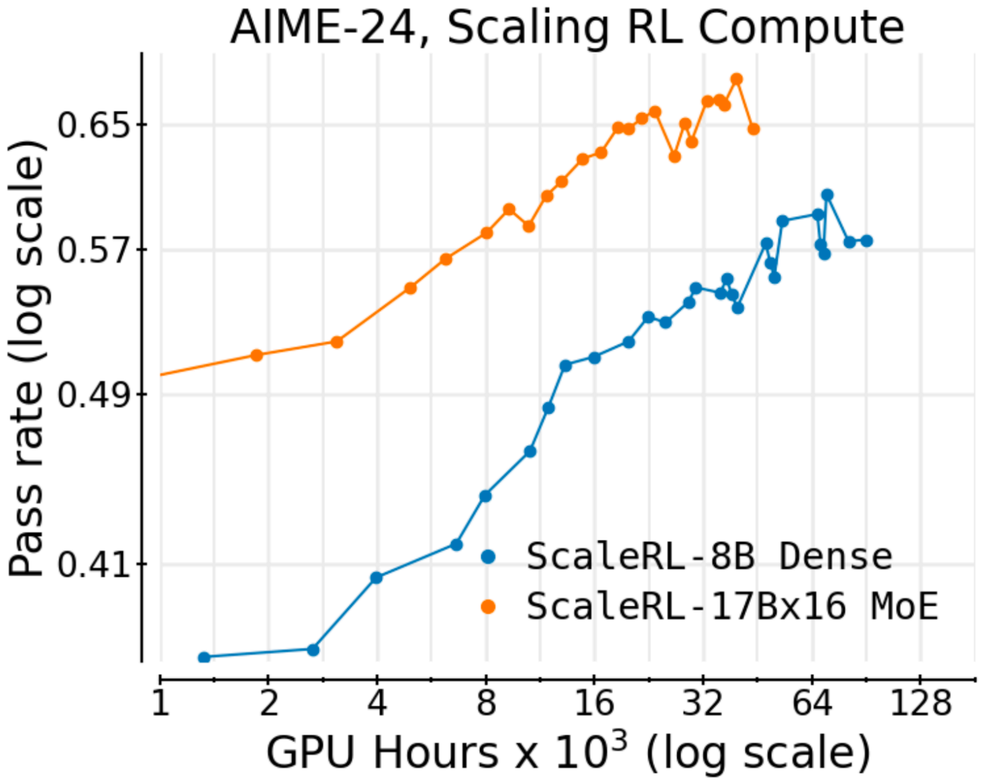
[1]Meta, [2]UT Austin, [3]UCL, [4]UC Berkeley, [5]Harvard University, [6]Periodic Labs
[*]Equal contribution, [†]Work done at Meta

Reinforcement learning (RL) has become central to training large language models (LLMs), yet the field lacks predictive scaling methodologies comparable to those established for pre-training. Despite rapidly rising compute budgets, there is no principled understanding of how to evaluate algorithmic improvements for scaling RL compute. We present the first large-scale systematic study, amounting to more than 400,000 GPU-hours, that defines a principled framework for analyzing and predicting RL scaling in LLMs. We fit sigmoidal compute-performance curves for RL training and ablate a wide range of common design choices to analyze their effects on asymptotic performance and compute efficiency. We observe: (1) Not all recipes yield similar asymptotic performance, (2) Details such as loss aggregation, normalization, curriculum, and off-policy algorithm primarily modulate compute efficiency without materially shifting the asymptote, and (3) Stable, scalable recipes follow predictable scaling trajectories, enabling extrapolation from smaller-scale runs. Combining these insights, we propose a *best-practice* recipe, **ScaleRL**, and demonstrate its effectiveness by successfully scaling and predicting validation performance on a single RL run scaled up to 100,000 GPU-hours. Our work provides both a *scientific framework* for analyzing scaling in RL and a practical recipe that brings RL training closer to the predictability long achieved in pre-training.

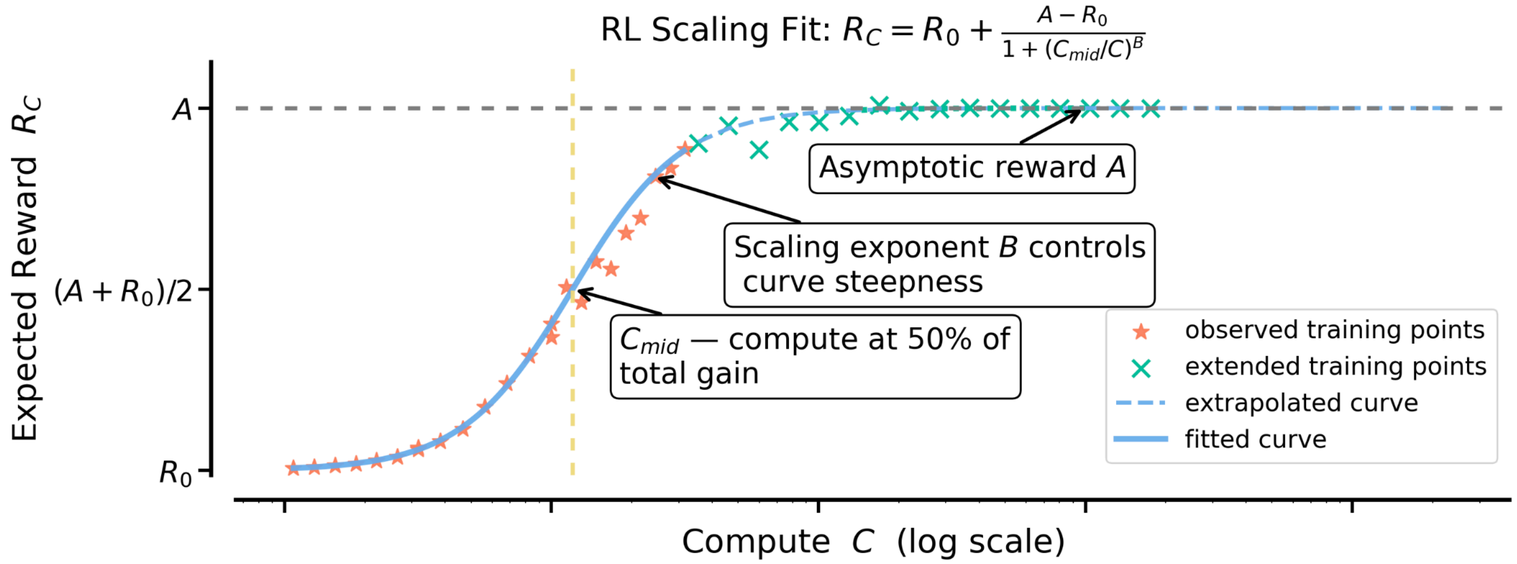
Correspondence: {lovish, brandfon}@meta.com, {devvrit.03, rishabhagarwal.467}@gmail.com

Meta





- Gains on the training distribution transfers to downstream tasks.

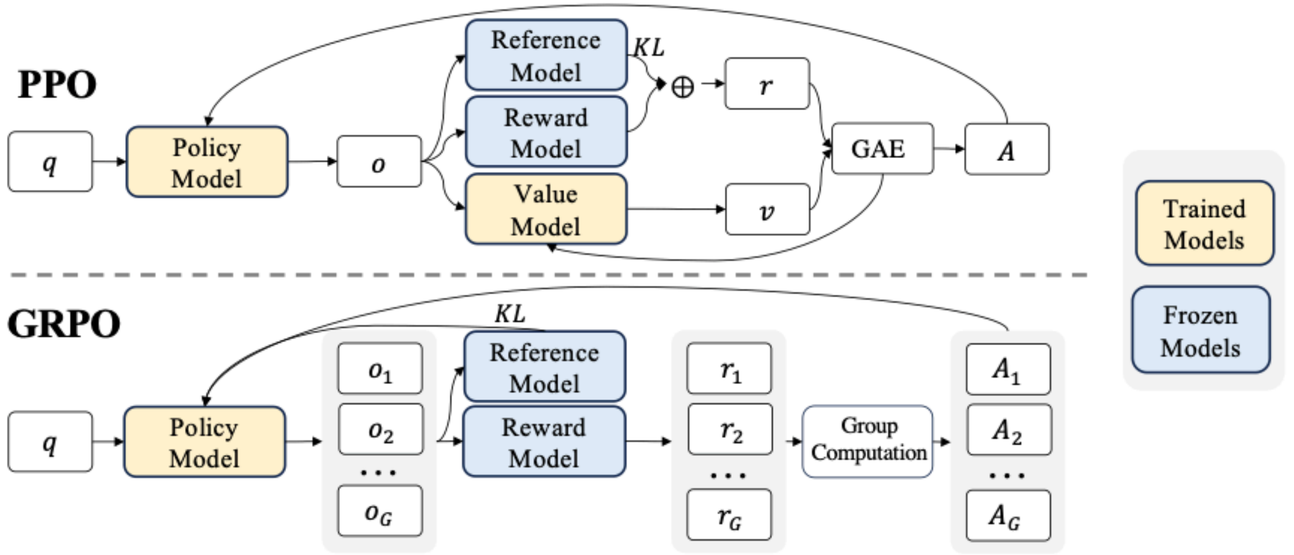


$$\overbrace{R_C - R_0}^{\text{Reward Gain}} = \overbrace{(A - R_0)}^{\text{Asymptotic Reward Gain}} \times \underbrace{\frac{1}{1 + (C_{\text{mid}}/C)^B}}_{\text{Compute Efficiency}}$$

(fixed model and training data)

- A is the asymptotic pass rate.
- B captures the compute efficiency.
- C_{mid} corresponds to the compute that would give half of the gains.

GRPO recap



$$\hat{A}_i = \frac{r_i - \text{mean}(\{r_j\}_{j=1}^{G})}{\text{std}(\{r_j\}_{j=1}^{G}) + \epsilon}$$

$$\rho_{i,t}(\theta) = \frac{\pi_{train}(y_{i,t} \mid x, y_{i,<t}, \theta)}{\pi_{gen}(y_{i,t} \mid x, y_{i,<t}, \theta_{\text{old}})}$$

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{\substack{x \sim D, \\ \{y_i\}_{i=1}^{G} \sim \pi_{\text{gen}}(\cdot \mid x, \theta_{\text{old}})}} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \min\left(\rho_{i,t}(\theta) \hat{A}_i,\ \text{clip}(\rho_{i,t}(\theta), 1 \pm \epsilon) \hat{A}_i \right) \right]$$

Algorithmic choices

- Loss: GRPO [Shao et al., ’24], DAPO [Yu et al., ’25], GSPO [Zheng et al., ’25], CISPO [Chen et al., ’25], …
  - Importance sampling at token level vs. sequence level.
  - Clipping details
- Compute precision: e.g., FP32 vs. FP16
- Advantage normalization: whether to compute standard deviation per prompt or per batch.
- Curriculum learning: in particular, filtering easy questions.

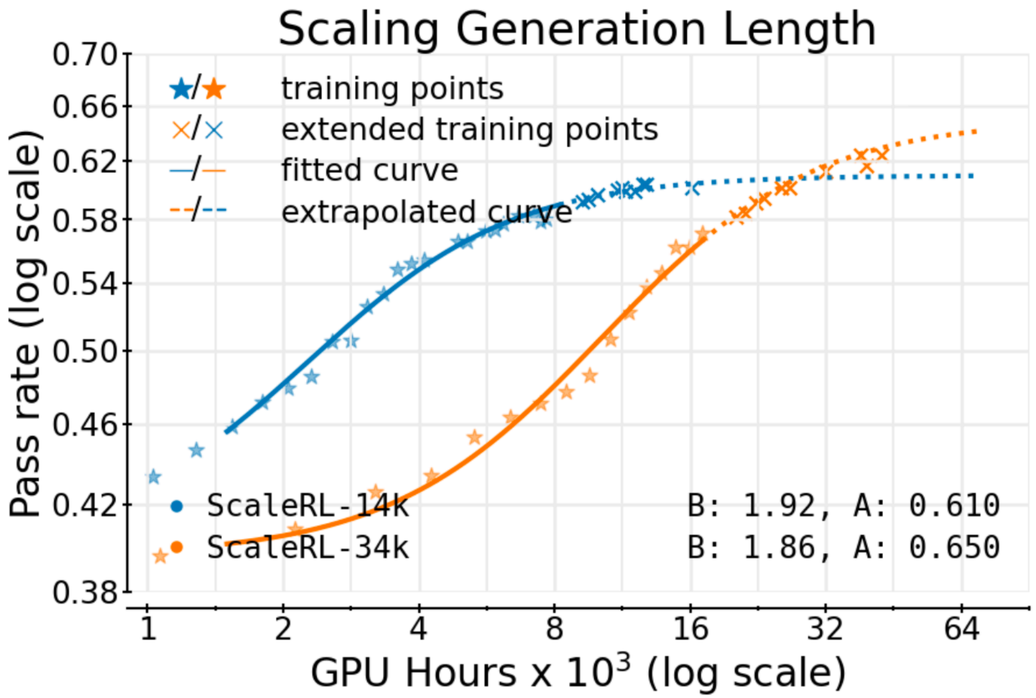
See [Khatri et al., ’25] for more details.

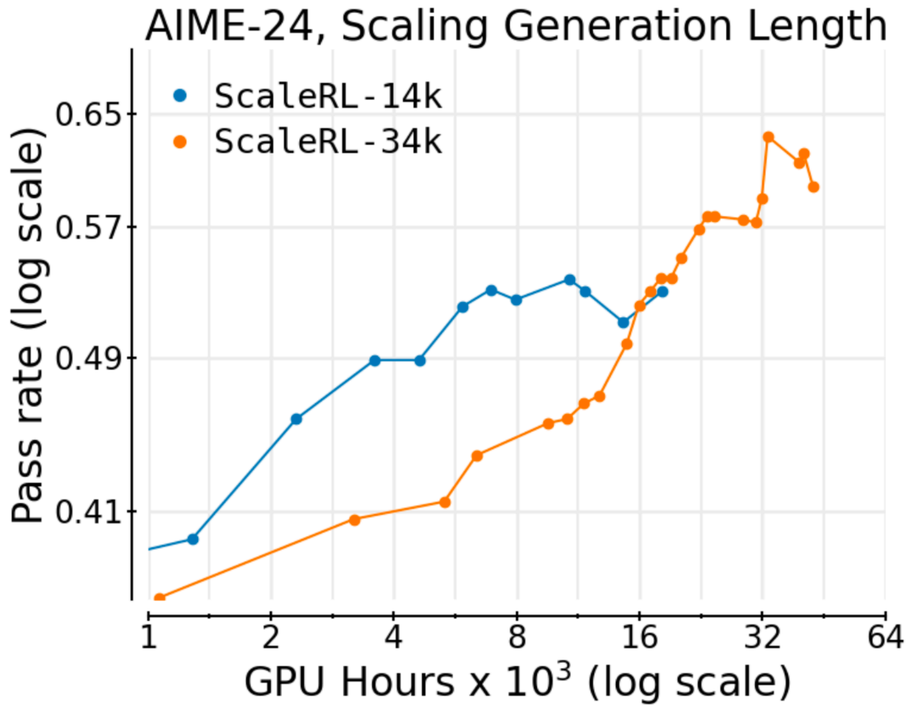
What to scale?

Main takeaways

- Larger models are (probably) better.
  - Better efficiency and better asymptotic limit.
- Longer generation length (context budget) is (probably) better.
  - Worse efficiency, better asymptotic limit.
- Larger global batch size (#prompts x #generations) is (probably) better.
  - Worse efficiency, better asymptotic limit.

“RL with longer generation budget is better.”





Larger generation length <—> More thinking per questions <—> Scaling at inference

Test-Time Scaling

Scaling inference (parallel)

with a verifier

- Assume your answers can be verified.
  - E.g., formal math or having test cases in coding.
- The probability of generating a correct answer is non-zero.
  - $\mathbb{P}(\mathrm{Ver}(y; x) = 1) = p > 0.$
  - In average, we would have a satisfying answer with p^{-1} generations.
- In some cases, we may be able to train a verifier.

Scaling inference (parallel)

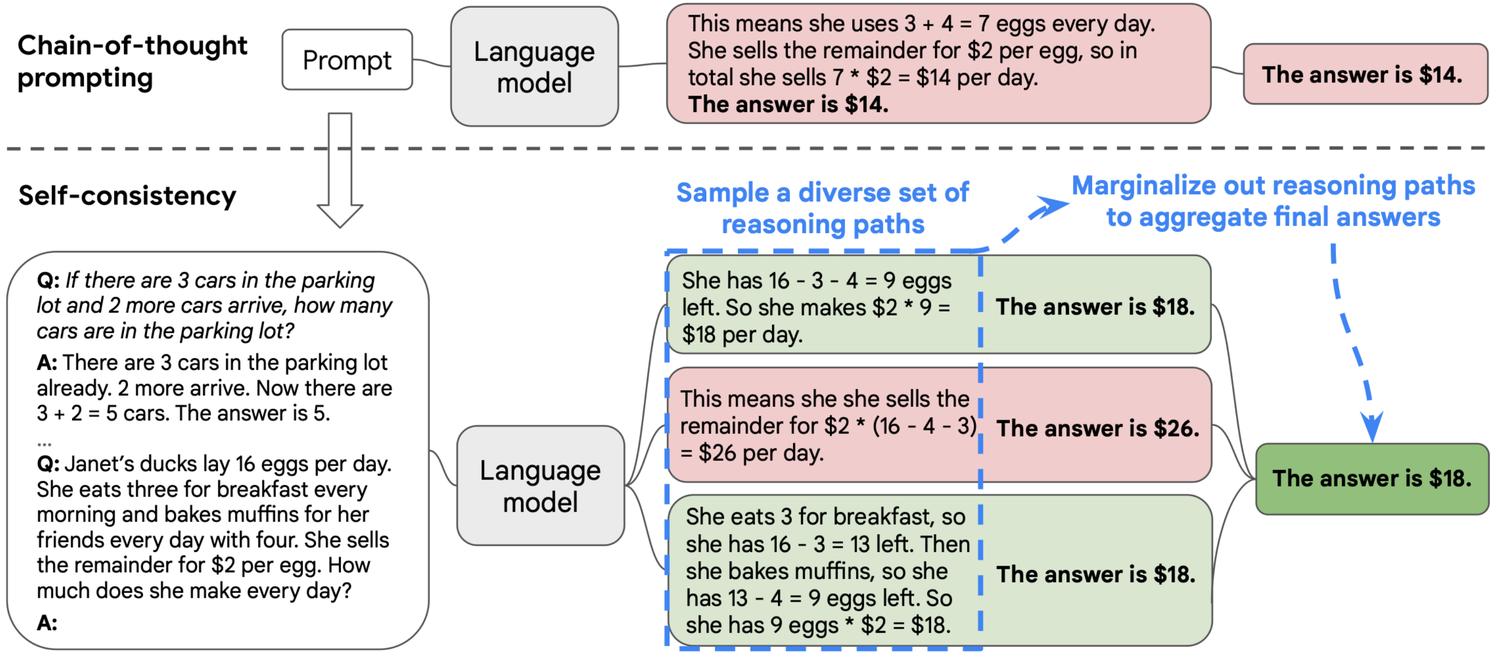
without a verifier

- Take a math questions with a final numerical answer.
  - We can't usually verify the answer easily.
- There are numerous ways to solve the problem correctly and they all reach the same answer.
- There are numerous ways to solve the problem incorrectly and reach different wrong answers.
- *We want to find the most probable final numerical answer.*
  - This is not necessarily the same as the answer given by the greedy decoding solution.

Scaling inference (parallel)

majority voting and self consistency [Wang et al. 2023]

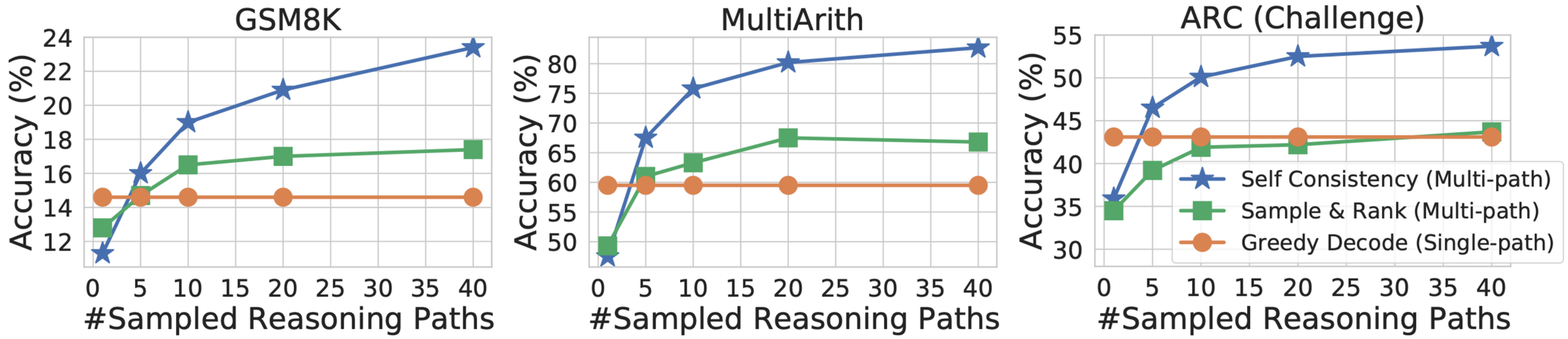
- We can sample N solutions and take the **majority vote** among the final answers.
  - More generally, finding the most consistent answer.



Scaling inference (parallel)

majority voting and self consistency [Wang et al. 2023]

- We can sample N solutions and take the majority vote among the final answers.
  - More generally, finding the most consistent answer.



Scaling inference (sequential)

- We can also teach the models to generally think longer.
  - Iterative refinement: 1) Plan and Reason, 2) Self-check, 3) Refine and repeat.
  - Teaching the model the ability to *backtrack* from mistakes.
- The model can learn such behavior through the context or training or …

s1

s1: Simple test-time scaling

Niklas Muennighoff [*1 3 4] **Zitong Yang** [*1] **Weijia Shi** [*2 3] **Xiang Lisa Li** [*1] **Li Fei-Fei** [1] **Hannaneh Hajishirzi** [2 3]
Luke Zettlemoyer [2] **Percy Liang** [1] **Emmanuel Candès** [1] **Tatsunori Hashimoto** [1]

- Budget forcing: forcing the model to think more or less (depending on the number of tokens).
- Assume that the model first *thinks* and then outputs the final solution and answer. —> goal: adjust thinking tokens.
- If the model is thinking below the threshold: we put “Wait” before it wants to terminate thinking.
- If the model is thinking too much, we interrupt it with “Final Answer:”

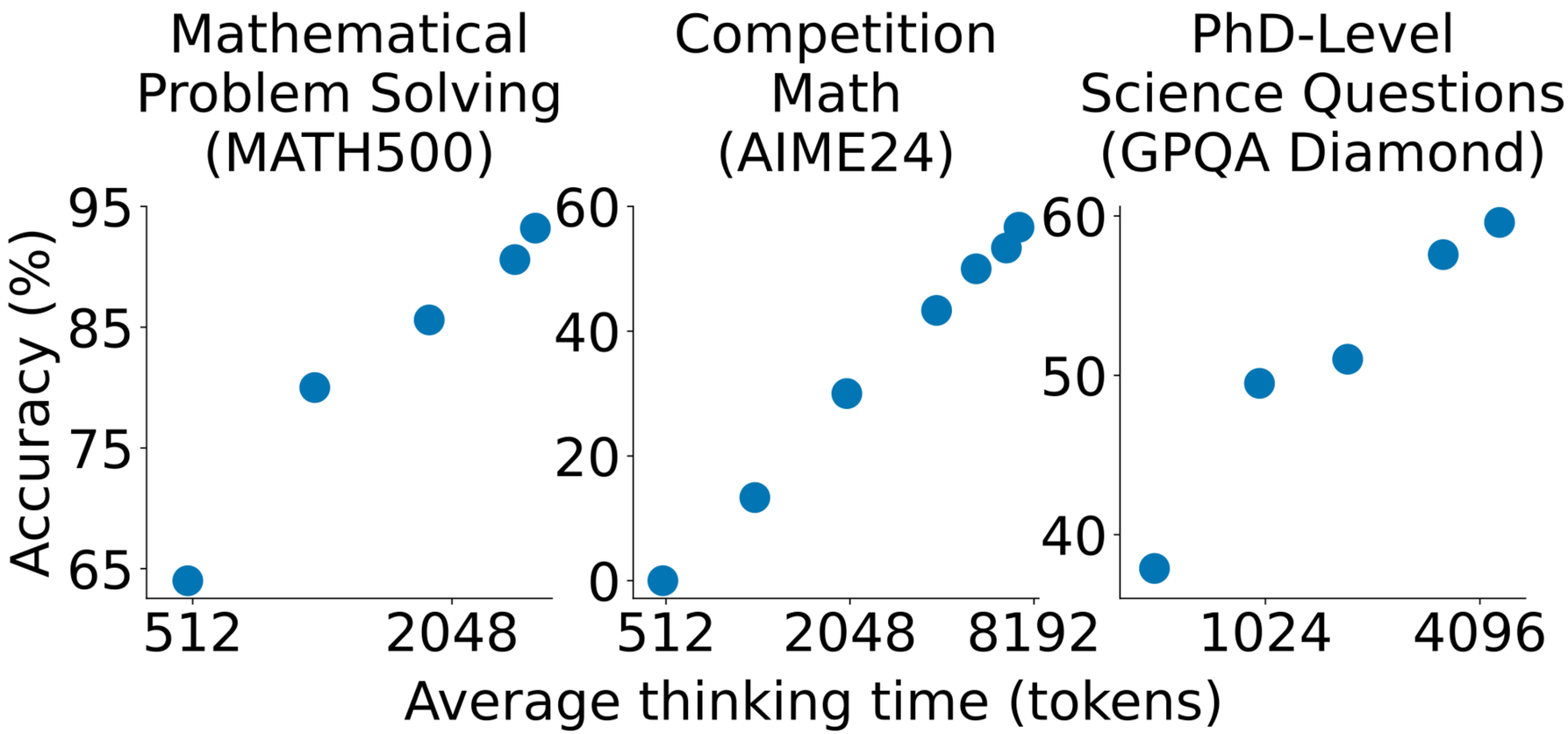
s1

How many r in raspberry? **Question**

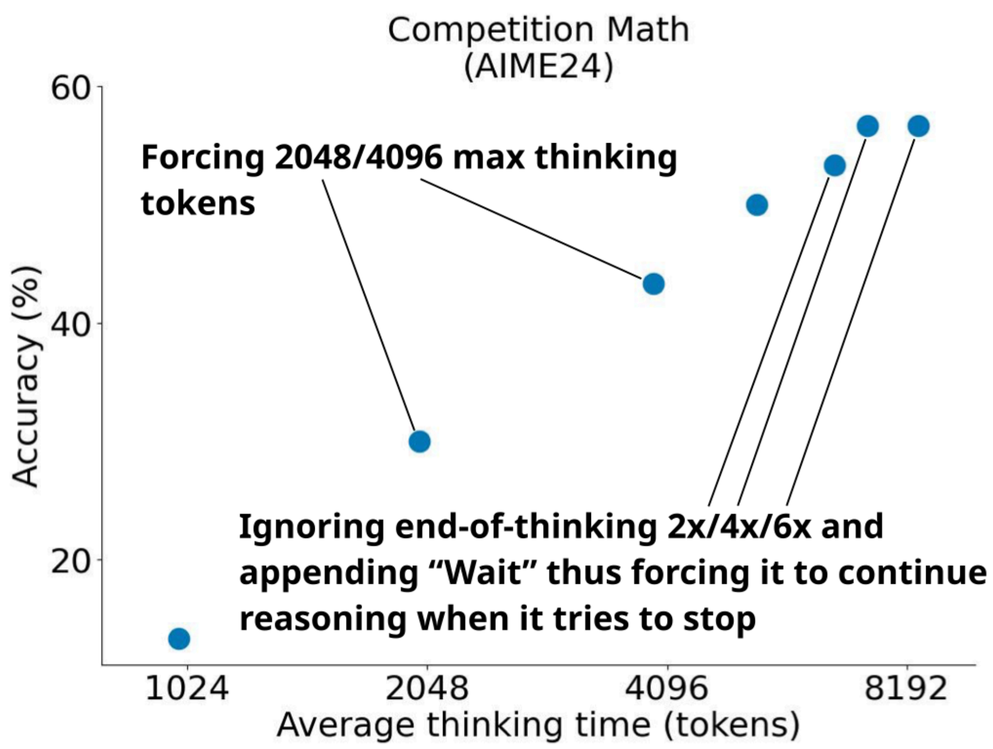
Let's break down the process of counting the letter 'r' in the word "raspberry" ...
* First letter: 'r' - This is an 'r', count = 1.
* Second letter: 'a' - Not an 'r', count remains 1 ...
* Sixth letter: 'e' - Not an 'r', count remains 1.
* Seventh letter: 'r' - This is an 'r', count = 2.
* Eighth letter: '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**

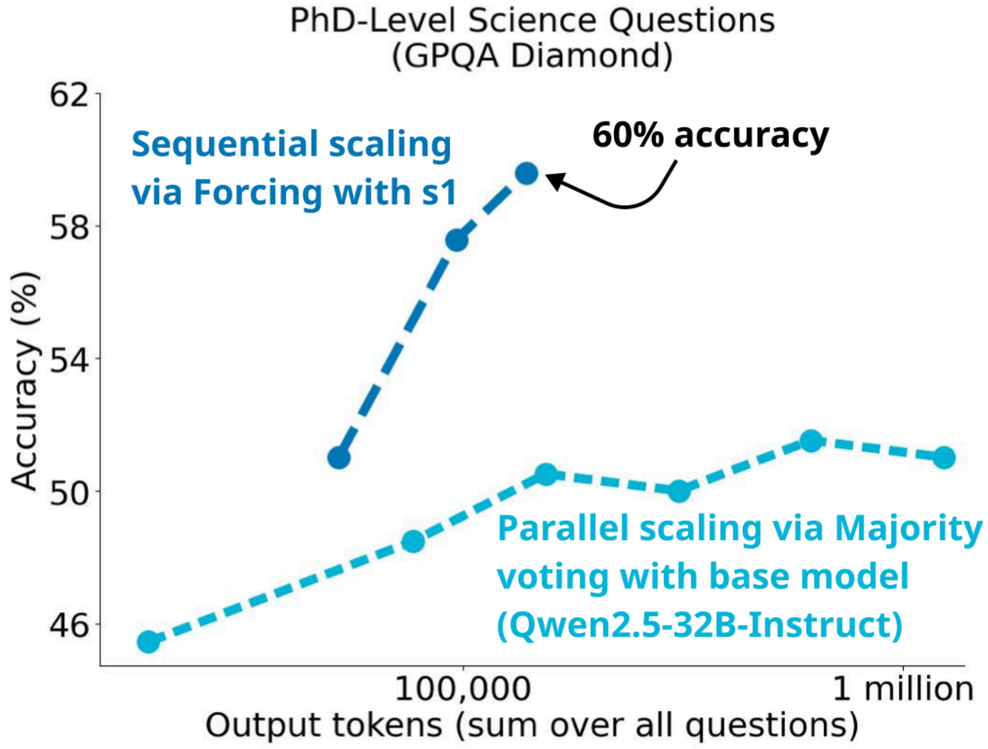
My initial answer of 2 was incorrect due to a quick reading of the word. **Final Answer:** The final answer is $\boxed{3}$ **Response**

s1


Mathematical Problem Solving (MATH500)
Competition Math (AIME24)
PhD-Level Science Questions (GPQA Diamond)
Accuracy (%)
95
85
75
65
60
40
20
0
60
50
40
512
2048
512
2048
8192
1024
4096
Average thinking time (tokens)

s1


Competition Math
(AIME24)
Accuracy (%)
60
40
20
Forcing 2048/4096 max thinking tokens
Ignoring end-of-thinking 2x/4x/6x and appending "Wait" thus forcing it to continue reasoning when it tries to stop
1024
2048
4096
8192
Average thinking time (tokens)


PhD-Level Science Questions
(GPQA Diamond)
Accuracy (%)
62
58
54
50
46
Sequential scaling via Forcing with s1
60% accuracy
Parallel scaling via Majority voting with base model (Qwen2.5-32B-Instruct)
100,000
1 million
Output tokens (sum over all questions)

Search approaches for test-time scaling

Usually for well-structured problems

Teaching search to LMs.

Stream of Search (SoS): Learning to Search in Language

Kanishk Gandhi* Stanford University
Denise Lee Stanford University
Gabriel Grand MIT
Muxin Liu Harvey Mudd
Winson Cheng Stanford University
Archit Sharma Stanford University
Noah D. Goodman Stanford University

Abstract

Language models are rarely shown fruitful mistakes while training. They then struggle to look beyond the next token, suffering from a snowballing of errors and struggling to predict the consequence of their actions several steps ahead. In this paper, we show how language models can be taught to search by representing the process of search in language, as a flattened string — a *stream of search (SoS)*. We propose a unified language for search that captures an array of different symbolic search strategies. We demonstrate our approach using the simple yet difficult game of Countdown, where the goal is to combine input numbers with arithmetic operations to reach a target number. We pretrain a transformer-based language model from scratch on a dataset of streams of search generated by heuristic solvers. We find that SoS pretraining increases search accuracy by 25% over models trained to predict only the optimal search trajectory. We further finetune this model with two policy improvement methods: Advantage-Induced Policy Alignment (APA) and Self-Taught Reasoner (STaR). The finetuned SoS models solve 36% of previously unsolved problems, including problems that cannot be solved by any of the heuristic solvers. Our results indicate that language models can learn to solve problems via search, self-improve to flexibly use different search strategies, and potentially discover new ones. [1]

"To err is human, to backtrack divine"
—Apocryphal

Monte Carlo Tree Search (MCTS)

RESEARCH

AlphaZero: Shedding new light on chess, shogi, and Go

6 DECEMBER 2018

David Silver, Thomas Hubert, Julian Schrittwieser, Demis Hassabis

Share

SCIENCE

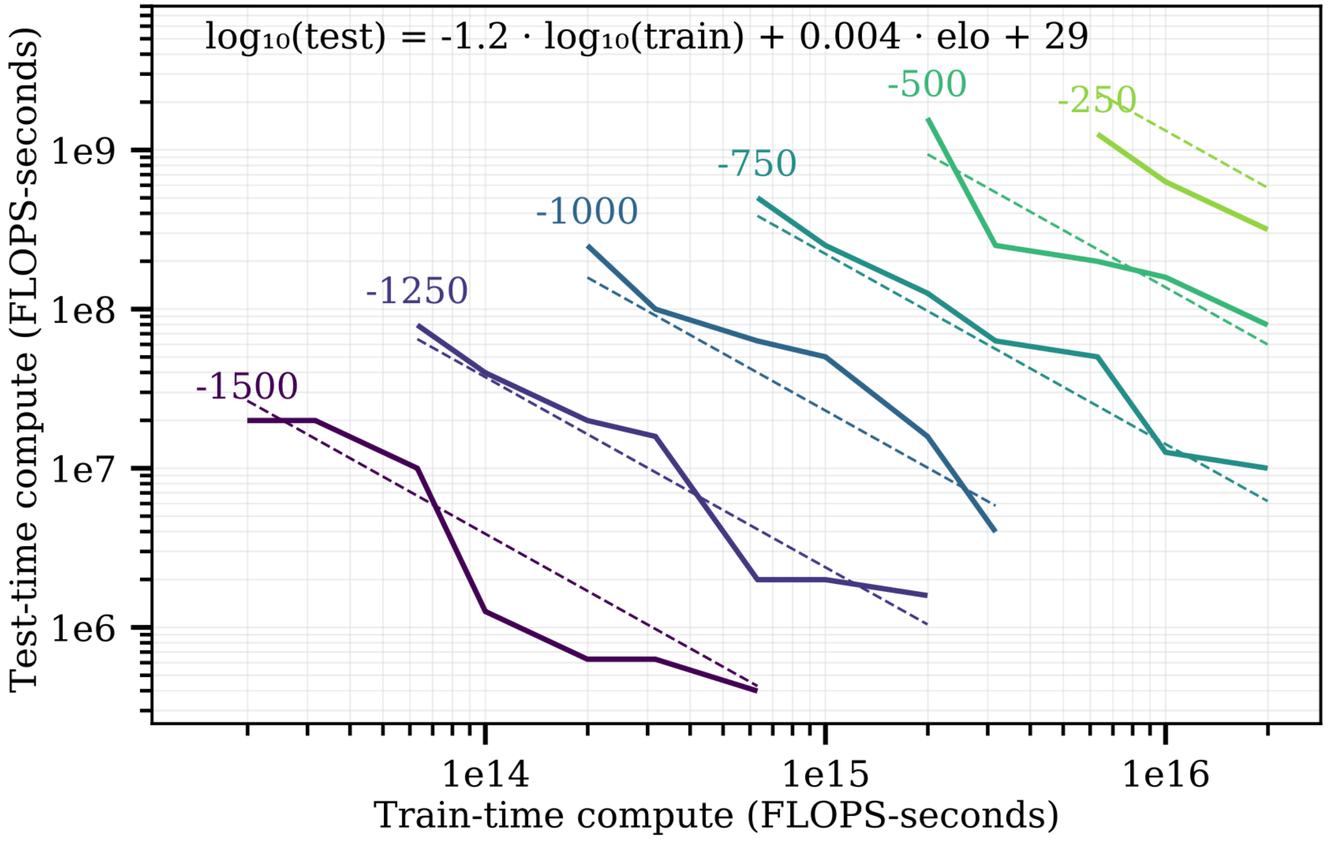
AI achieves silver-medal standard solving International Mathematical Olympiad problems

25 JULY 2024

AlphaProof and AlphaGeometry teams

Test-time vs. train-time scaling

For board games

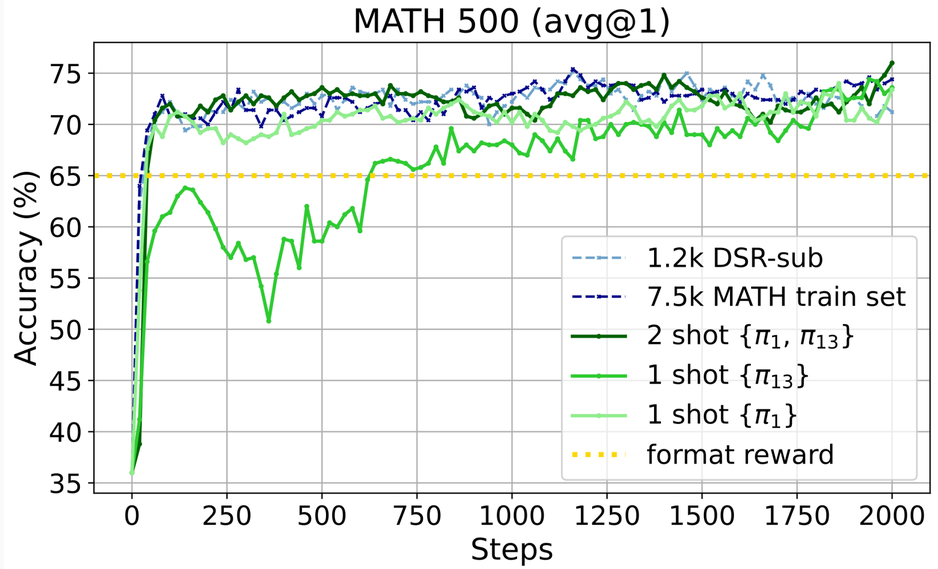


Scaling Scaling Laws with Board Games

Andy L. Jones

Relation between RL and test-time scaling

- There are single samples such that RL on those individual samples yields similar performance to doing the RL on the entire dataset!



- Why?
- Model learns self-reflective behaviors by RL on a single (suitable) sample.

Reinforcement Learning for Reasoning in Large Language Models with *One* Training Example

Yiping Wang[1 † *] Qing Yang[2] Zhiyuan Zeng[1] Liliang Ren[3] Liyuan Liu[3]
Baolin Peng[3] Hao Cheng[3] Xuehai He[4] Kuan Wang[5] Jianfeng Gao[3]
Weizhu Chen[3] Shuohang Wang[3 †] Simon Shaolei Du[1 †] Yelong Shen[3 †]