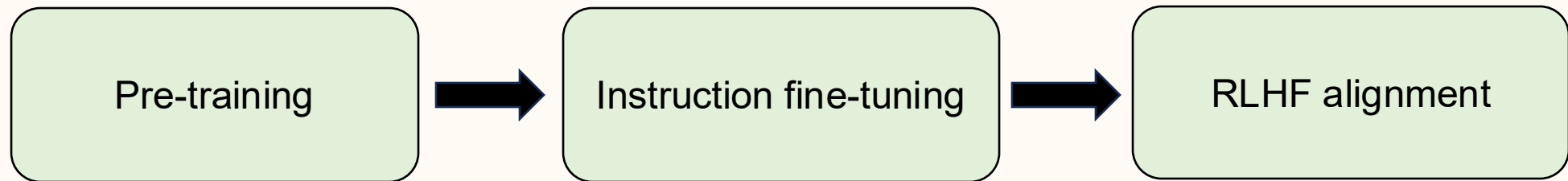


# Chapter 3: Reinforcement Learning of Large Language Models

Ernest K. Ryu

University of California, Los Angeles

# 3-stage training of LLMs



With pre-training and instruction fine-tuning, the language model  $\pi_\theta$  is able to generate language and follow instructions.

RLHF further aligns LLM with human values and expectations.

# Why RLHF?

Pre-training and supervised instruction fine-tuning use the next-token-prediction loss. The dataset presents a correct answer and forces the model to imitate it, like imitation learning.

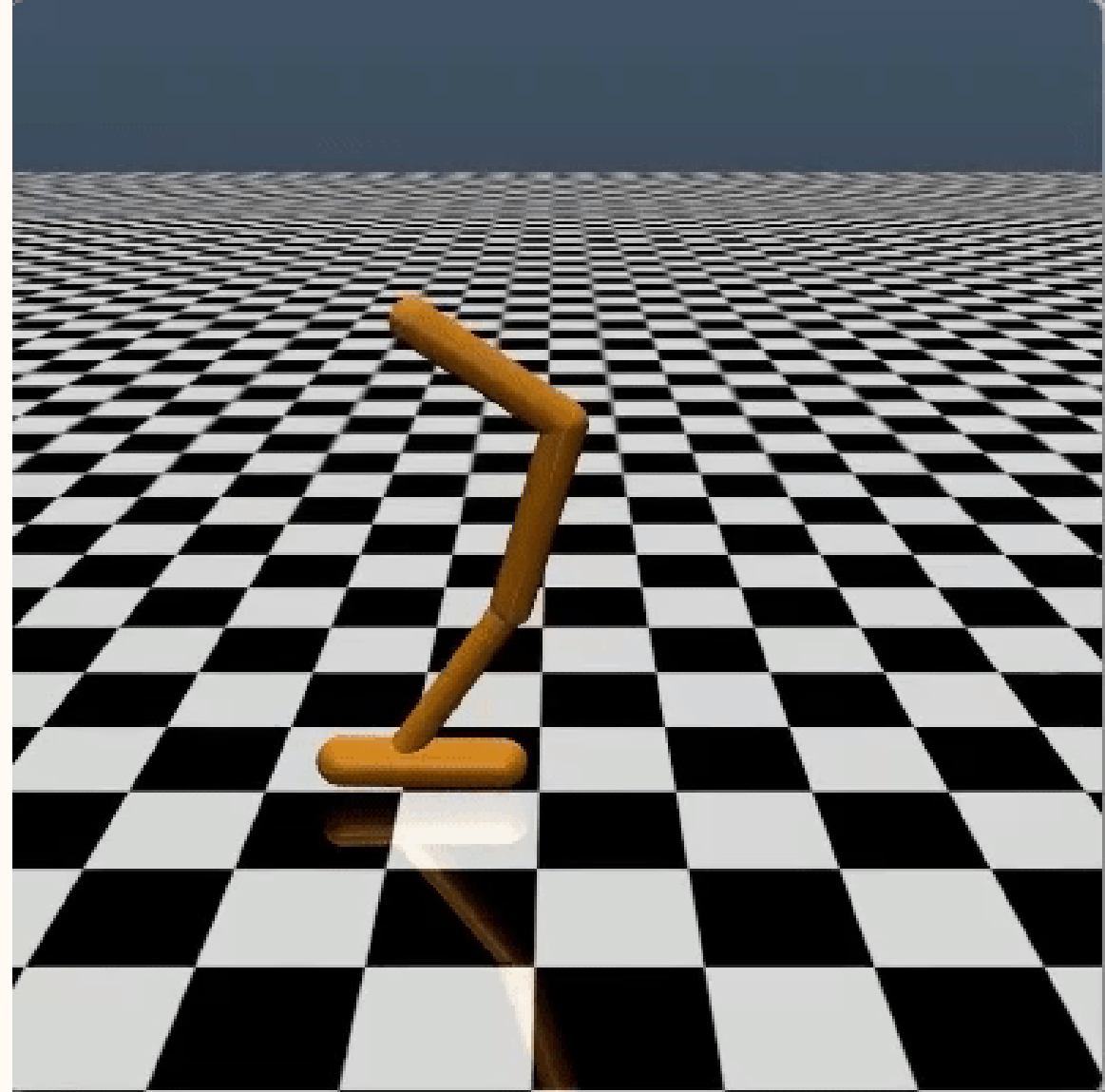
- Large language models can generate outputs that are untruthful, toxic, or simply not helpful to the user. Next token prediction does not provide an effective way to steer a model away from bad outputs.
  - The pre-training dataset contains some data that is unkind, so a model trained with next-token-prediction may sometimes be unkind to the user. How do we explicitly tell the model to be kind to the user?
- Next-token-prediction is not appropriate for specifying abstract goals.
  - E.g. “Follow the user’s instructions helpfully and safely.”
  - E.g. “Refuse a user’s command if it is unethical or dangerous.”

# RLHF from RL

Reinforcement learning (RL), aims to control an agent to achieve high “reward”, but this reward is sometimes difficult to specify as a formal function.

Example) We know a backflip when we see it, but it is difficult program a function that returns positive reward upon a successful backflip.

RL with human feedback (RLHF) uses human feedback to determine the desired behavior, often by training a *reward model*.



<https://openai.com/index/learning-from-human-preferences/>

# Aligning LLMs with RLHF

In the InstructGPT paper, RLHF is carried out with three neural networks.

- $\pi_\theta(u_{\ell+1}|u_1, \dots, u_\ell)$ : Instruction fine-tuned LM, 175B GPT-3.
- $r_\psi$ : Reward model (RM), initialized from a pre-trained LM, 6B GPT-3 + new head to make the output a scalar.
- $V_\phi$ : Value function model (baseline function for PPO), initialized to be RM.

(Smaller 6B RM was used because with a 175B RM, (1) training was more unstable for some reason, and (2) using a 175B RM and value function greatly increase the compute requirements of PPO.)

- Question to think about: Is  $r_\psi$  and  $V_\phi$  really necessary?

# Reward model: Training data

Let  $x$  be a prompt specifying a task, and  $x$  requires an answer.

The instruction fine-tuned model generates  $K$ -independent completions  $y_1, \dots, y_K$ .

- For the completions to be distinct, greedy sampling or beam search should not be used. It may even help to large temperature ( $\beta = 1$  or higher) to increase the randomness in the response.
- $K = 4$  to 9 in InstructGPT paper.

Human annotator is given detailed criterion and (human) training. Then, the  $y_i > y_j$  or  $y_i < y_j$  annotations for all pairs  $(y_i, y_j)$  are provided. ( $K$  choose 2 comparison annotations.)

# Reward model: Bradley–Terry

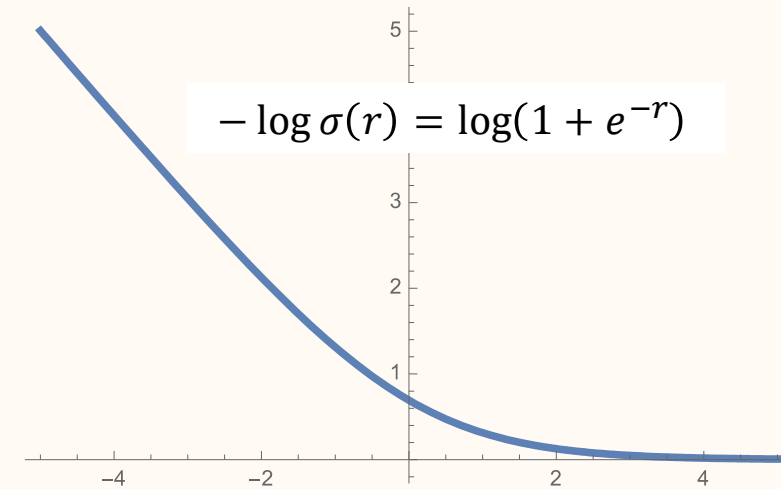
Reward model  $r_\psi$  is trained as a Bradley–Terry model: Minimizing

$$\mathcal{L}(\psi) = \mathbb{E}_{\substack{(x, y_{\text{win}}, y_{\text{lose}}) \\ y_{\text{win}} > y_{\text{lose}}}} [-\log \sigma(r_\psi(x, y_{\text{win}}) - r_\psi(x, y_{\text{lose}}))] = \mathbb{E}_{\substack{(x, y_{\text{win}}, y_{\text{lose}}) \\ y_{\text{win}} > y_{\text{lose}}}} \left[ -\log \left( \frac{e^{r_\psi(x, y_{\text{win}})}}{e^{r_\psi(x, y_{\text{win}})} + e^{r_\psi(x, y_{\text{lose}})}} \right) \right]$$

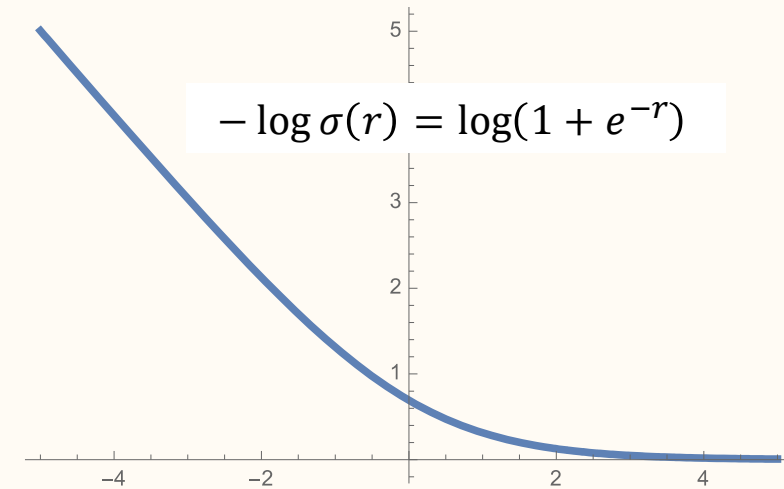
where  $y_{\text{win}} > y_{\text{lose}}$  according to human preference.

Given human annotation data  $\mathcal{D}$  containing  $(x, y_1, \dots, y_K)$  and the human annotation specifying  $y_i > y_j$  for all  $K$  choose 2 pairs, train the reward model  $r_\psi$  by minimizing

$$\mathcal{L}(\psi) \approx \sum_{\substack{(x, y_i, y_j) \in \mathcal{D} \\ y_i > y_j}} -\log \sigma(r_\psi(x, y_i) - r_\psi(x, y_j))$$



# Reward model: Bradley–Terry



$$\mathcal{L}(\psi) = \mathbb{E}_{\substack{(x, y_{\text{win}}, y_{\text{lose}}) \\ y_{\text{win}} > y_{\text{lose}}}} [-\log \sigma(r_{\psi}(x, y_{\text{win}}) - r_{\psi}(x, y_{\text{lose}}))] = \mathbb{E}_{\substack{(x, y_{\text{win}}, y_{\text{lose}}) \\ y_{\text{win}} > y_{\text{lose}}}} \left[ -\log \left( \frac{e^{r_{\psi}(x, y_{\text{win}})}}{e^{r_{\psi}(x, y_{\text{win}})} + e^{r_{\psi}(x, y_{\text{lose}})}} \right) \right]$$

$r_{\psi}$  will be trained such that  $r_{\psi}(x, y) \gg$  (average) is  $y$  is a “good” completion and vice versa.

Note, this can be viewed soft-max regression with  $K = 2$  (logistic regression) on determining probability of the two events:  $[(x, y_1) \text{ is better}]$  vs.  $[(x, y_2) \text{ is better}]$ .



# Best-of-N sampling

Given a trained reward model  $r_\psi$ , how do we use it to generate high-reward completions?

Best-of-N sampling:

1. Generate  $N$  text outputs.
2. Select the best one as determined by the reward model  $r_\psi$ .

Advantage: Simple and effective<sup>#</sup> way to utilize a reward model trained from human feedback. Also, no need for RL training, which can be tricky. (Best-of-N should always be used as a baseline approach.)

Downside: Sampling requires  $N$  generations, so inefficient. (If you are willing to pay the cost of multiple generations, you can view best-of-N as an instance of test-time-scaling.)

<sup>#</sup>L. Gao, J. Schulman, and J. Hilton, Scaling laws for reward model overoptimization, *ICML*, 2023.

A. Beirami, A. Agarwal, J. Berant, A. D'Amour, J. Eisenstein, C. Nagpal, and A. T. Suresh, Theoretical guarantees on the best-of-n alignment policy, *ICML*, 2025.

# The RL of RLHF

RL with human feedback (RLHF) on LLMs further fine-tunes the LLM  $\pi_\theta$  so that the completion achieves high reward as measured by the reward model  $r_\psi$ .

What is the MDP?

- LLMs are *autoregressive* models and are decidedly not Markovian. (Next token generation depends on the *entire* past, not just the previous token.) However, any non-Markovian process can be made “Markovian” by defining the entire history as the “previous state.” In standard RL, the policy has the form  $\pi(a_t|s_t)$ , but in LLMs, we need something like  $\pi(a_t|s_1, \dots, s_t)$ .
- The “state” is the user prompt and the tokens generated so far  $u_1, \dots, u_\ell$ .
- Each action is the generation of one token. The policy is random, but the transition dynamics it deterministic, i.e., (current state, current action) = (next state).

# The RL of RLHF

## The RL setup

- Each timestep is a BPE token.
- The LLM  $\pi_{\theta}(u_{\ell+1}|u_1, \dots, u_{\ell})$  is our policy mapping current state  $(u_1, \dots, u_{\ell})$  to a distribution on the action (next token)  $u_{\ell+1}$ .
- Response generation is an episode, and an episode terminates when LM generates  $\langle \text{EOS} \rangle$ .
- No discount used, i.e., discount factor  $\gamma = 1$  is used.
- Reward (by reward model  $r_{\psi}$ ) is only provided at the end of the episode. There are no intermediate reward. Called “contextual bandit” setting.
- Sampling temperature  $\beta = 1$ .

# Proximal policy optimization (PPO)

- Let  $\pi_\theta(u_{\ell+1}|u_1, \dots, u_\ell)$  be our LLM and the RL policy.
- Let  $x$  be a text prompt and  $y = y_{1:T+1}$  be its completion by  $\pi_\theta$ . (So  $y_{T+1} = \langle \text{EOS} \rangle$ .)
- Let  $y_{1:t}$  the partial completion up to token  $t$ .
- The PPO-clip ratio is set to  $\varepsilon = 0.2$ .

PPO maintains a value function model  $V_\phi(x, y_{1:t})$ : Given  $(x, y_{1:t})$ , what is the expected reward if we continue generation with  $\pi_\theta$ .

Advantage  $\hat{A} = r_\psi(x, y_{1:T+1}) - V_\phi(x, y_{1:t})$ : How good is the total completion  $y_{t+1:T+1}$  compared to what  $V_\phi$  was expecting based on  $y_{1:t}$ ?

# Proximal policy optimization (PPO)

If  $\hat{A} = r_\psi(x, y_{1:T+1}) - V_\phi(x, y_{1:t}) > 0$ , then  $y_{t+1:T+1}$  was a good completion. We should adjust  $\pi_\theta$  to make those actions more likely.

If  $\hat{A} = r_\psi(x, y_{1:T+1}) - V_\phi(x, y_{1:t}) < 0$ , then  $y_{t+1:T+1}$  was a bad completion. We should adjust  $\pi_\theta$  to make make those actions less likely.

(Actually GAE was used for  $\hat{A}$ , but let's consider the simpler advantage estimate for simplicity.)

# PPO v.0 (susceptible to over-optimization)

while (not converged)

sample  $N$  trajectories  $(x^{(i)}, y_{1:T^{(i)}+1}^{(i)}) \sim (p_0, \pi_{\theta_{\text{curr}}}^{\text{RL}})$  for  $i = 1, \dots, N$

$\hat{A}_t^{(i)} = r_\psi(x^{(i)}, y_{1:T^{(i)}+1}^{(i)}) - V_\phi(x^{(i)}, y_{1:t}^{(i)})$  for  $i = 1, \dots, N, t = 0, \dots, T^{(i)}$

solve:

$$\underset{\theta_{\text{next}} \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^N \sum_{t=0}^{T^{(i)}} \mathcal{C}_\varepsilon \left( \frac{\pi_{\theta_{\text{next}}}^{\text{RL}}(y_{t+1}^{(i)} | x^{(i)}, y_{1:t}^{(i)})}{\pi_{\theta_{\text{curr}}}^{\text{RL}}(y_{t+1}^{(i)} | x^{(i)}, y_{1:t}^{(i)})}, \hat{A}_t^{(i)} \right)$$

$\theta_{\text{curr}} = \theta_{\text{next}}$

solve:

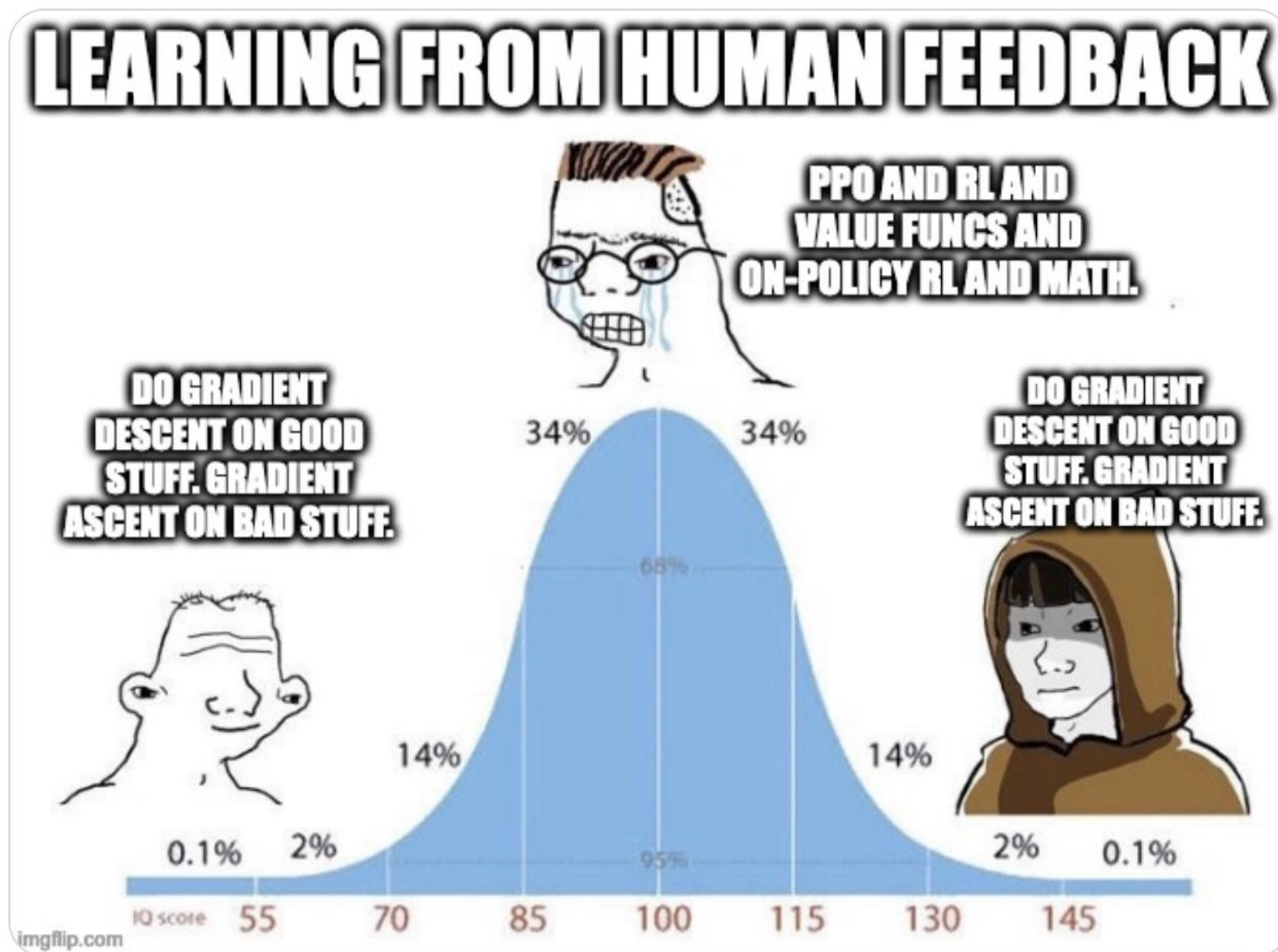
$$\mathcal{C}_\varepsilon(\ell, A) = \min(\ell A, \text{clip}_{1-\varepsilon}^{1+\varepsilon}(\ell) A)$$

$$\underset{\phi \in \mathbb{R}^q}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^N \frac{1}{T^{(i)}} \sum_{t=0}^{T^{(i)}} \frac{1}{2} (r_\psi(x^{(i)}, y_{1:T^{(i)}+1}^{(i)}) - V_\phi(x^{(i)}, y_{1:t}^{(i)}))^2$$

end

Tom Goldstein  
@tomgoldsteincs

...



# Over-optimization

Goodhart's law: When a measure becomes a target, it ceases to be a good measure.

Problem with PPO v.0: Over-optimization.

- Reward model is imperfect, so we should not overfit to it.
  - Maximizing reward model too much will result in adversarial generation that seems good to the reward model by exploiting the imperfections of the model.
  - Reward model  $r_\psi$  was trained on  $\pi^{\text{SFT}}$ , the supervised-fine-tuned (pre-trained and instruction fine-tuned) LLM. So  $r_\psi$  is informative about responses generated by  $\pi_\theta^{\text{RL}}$  only when  $\pi_\theta^{\text{RL}}$  is “close” to  $\pi^{\text{SFT}}$ . ( $\pi_\theta^{\text{RL}}$  is initialized to  $\pi^{\text{SFT}}$ .)
- Moving away from the pre-trained and instruction-fine tuned model too much will cause the language model to lose its main capabilities.
  - Fine-tuning too much can break the model, causing it to output nonsense tokens.



# Over-optimization

Overfitting concerns fitting data, while over-optimization concerns fitting the reward model. Both lead to poor generalization, but there are some substantive differences.

Over-optimization leads to the model learning adversarial examples, also referred to as *reward hacking*.

PPO's clipped objective prevents over-optimization in the sense of keeping  $\theta_{\text{next}}$  close to  $\theta_{\text{curr}}$ . This is different.

Resolution) Impose a KL-divergence penalty term, ensuring that  $\pi_{\theta}^{\text{RL}}$  is “close” to  $\pi^{\text{SFT}}$ .

# KL-penalty and pre-training loss

RLHF with KL-penalty maximizes the objective:

$$\begin{aligned}\mathcal{J}(\theta) &= \mathbb{E}_{(x,y) \sim D_{\pi_{\theta}^{\text{RL}}}} \left[ r_{\psi}(x, y) \right] - \beta \mathbb{E}_{x \sim \mathcal{D}} \left[ D_{\text{KL}}(\pi_{\theta}^{\text{RL}}(\cdot | x) \| \pi^{\text{SFT}}(\cdot | x)) \right] \\ &= \mathbb{E}_{(x,y) \sim D_{\pi_{\theta}^{\text{RL}}}} \left[ r_{\psi}(x, y) - \beta \log \frac{\pi_{\theta}^{\text{RL}}(y | x)}{\pi^{\text{SFT}}(y | x)} \right] \\ &= \mathbb{E}_{(x,y) \sim D_{\pi_{\theta}^{\text{RL}}}} \left[ r_{\psi}(x, y) - \beta \sum_{t=0}^T \log \frac{\pi_{\theta}^{\text{RL}}(y_{t+1} | x, y_{1:t})}{\pi^{\text{SFT}}(y_{t+1} | x, y_{1:t})} \right]\end{aligned}$$

where  $\beta > 0$ . The KL-penalty  $D_{\text{KL}}(\pi_{\theta}^{\text{RL}}(\cdot | x) \| \pi^{\text{SFT}}(\cdot | x))$  encourages  $\pi_{\theta}^{\text{RL}}$  is to stay close to  $\pi^{\text{SFT}}$ .

# KL-penalty and pre-training loss

$$\mathcal{J}(\theta) = \mathbb{E}_{(x,y) \sim D_{\pi_{\theta}^{\text{RL}}}} \left[ r_{\psi}(x, y) - \beta \sum_{t=0}^T \log \frac{\pi_{\theta}^{\text{RL}}(y_{t+1} \mid x, y_{1:t})}{\pi^{\text{SFT}}(y_{t+1} \mid x, y_{1:t})} \right]$$

Maximizing  $\mathcal{J}(\theta)$  can be thought of as equivalent to performing RL without an explicit penalty on an MDP with the same transition dynamics but with the modified rewards  $r_0, r_1, \dots, r_T$  given by

$$r_t = -\beta \log \frac{\pi_{\theta}^{\text{RL}}(y_{t+1} \mid x, y_{1:t})}{\pi^{\text{SFT}}(y_{t+1} \mid x, y_{1:t})}, \quad t = 0, \dots, T-1$$

$$r_T = r_{\psi}(x, y_{1:T+1}) - \beta \log \frac{\pi_{\theta}^{\text{RL}}(y_{T+1} \mid x, y_{1:T})}{\pi^{\text{SFT}}(y_{T+1} \mid x, y_{1:T})}$$

In other words, we can absorb the KL penalty into the rewards.

- Now the modified MDP has intermediate rewards.
- To clarify, our LLM notation has the episode terminating on the  $(T+1)$ -th step, while our previous RL notation had episodes terminating on the  $T$ -th step.
- The reward itself now depend on the parameter  $\theta$ , but if you redo the derivation, everything turns out okay.

# KL-penalty and pre-training loss

However, even with the KL-penalty, the base language modeling capability is damaged. Therefore, continue with the language model training during RLHF training by maximizing

$$\mathcal{J}(\theta) = \mathbb{E}_{(x,y) \sim D_{\pi_{\theta}^{\text{RL}}}} \left[ r_{\psi}(x,y) - \beta \log \left( \pi_{\theta}^{\text{RL}}(y | x) / \pi^{\text{SFT}}(y | x) \right) \right] + \eta \mathbb{E}_{x \sim D_{\text{pretrain}}} \left[ \log \left( \pi_{\theta}^{\text{RL}}(x) \right) \right]$$

The  $\eta$ -term is the next-token prediction loss used in pre-training, where  $\eta > 0$ .

PPO and pre-training update performed simultaneously or in alternating fashion.

# PPO with KL penalty

while (not converged)

sample  $N$  trajectories  $(x^{(i)}, y_{1:T^{(i)}+1}^{(i)}) \sim (p_0, \pi_{\theta_{\text{curr}}^{\text{RL}}}^{\text{RL}})$  for  $i = 1, \dots, N$

$$\hat{A}_t^{(i)} = r_\psi(x^{(i)}, y_{1:T^{(i)}+1}^{(i)}) - \underbrace{\beta \sum_{s=t}^{T^{(i)}} \log \left( \frac{\pi_{\theta_{\text{next}}^{\text{RL}}}^{\text{RL}}(y_{s+1}^{(i)} | x^{(i)}, y_{1:s}^{(i)})}{\pi_{\theta_{\text{curr}}^{\text{SFT}}}^{\text{SFT}}(y_{s+1}^{(i)} | x^{(i)}, y_{1:s}^{(i)})} \right)}_{=\hat{G}_t^{(i)}} - V_\phi(x^{(i)}, y_{1:t}^{(i)}) \quad \text{for } \begin{matrix} i=1, \dots, N \\ t=0, \dots, T^{(i)} \end{matrix}$$

solve:

$$\underset{\theta_{\text{next}} \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^N \sum_{t=0}^{T^{(i)}} \mathcal{C}_\varepsilon \left( \frac{\pi_{\theta_{\text{next}}^{\text{RL}}}^{\text{RL}}(y_{t+1}^{(i)} | x^{(i)}, y_{1:t}^{(i)})}{\pi_{\theta_{\text{curr}}^{\text{RL}}}^{\text{RL}}(y_{t+1}^{(i)} | x^{(i)}, y_{1:t}^{(i)})}, \hat{A}_t^{(i)} \right)$$

$$\theta_{\text{curr}} = \theta_{\text{next}}$$

solve:

$$\mathcal{C}_\varepsilon(\ell, A) = \min(\ell A, \text{clip}_{1-\varepsilon}^{1+\varepsilon}(\ell) A)$$

$$\underset{\phi \in \mathbb{R}^q}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^N \frac{1}{T^{(i)}} \sum_{t=0}^{T^{(i)}} \frac{1}{2} (\hat{G}_t^{(i)} - V_\phi(x^{(i)}, y_{1:t}^{(i)}))^2$$

end

# Closed-form solution of KL-regularized RL

Interestingly, there is a closed-form solution for the optimal KL-regularized policy given the reward model  $r = r_\psi$ . Ignoring the pre-training loss, the KL-regularized RL solves

$$\underset{\theta}{\text{maximize}} \quad \mathcal{J}(\pi_\theta; r) = \mathbb{E}_{\substack{x \sim \mathcal{D} \\ y \sim \pi_\theta}} \left[ r(x, y) - \beta \log \frac{\pi_\theta(y | x)}{\pi^{\text{SFT}}(y | x)} \right]$$

We can transform the loss as

$$\begin{aligned} -\frac{1}{\beta} \mathcal{J}(\pi_\theta; r) &= \mathbb{E}_{\substack{x \sim \mathcal{D} \\ y \sim \pi_\theta}} \left[ \log \frac{\pi_\theta(y | x)}{\pi^{\text{SFT}}(y | x)} - \frac{1}{\beta} r(x, y) \right] \\ &= \mathbb{E}_{\substack{x \sim \mathcal{D} \\ y \sim \pi_\theta}} \left[ \log \frac{\pi_\theta(y | x)}{\pi^{\text{SFT}}(y | x)} - \log \exp \left( \frac{1}{\beta} r(x, y) \right) + \log Z(x) - \log Z(x) \right] \\ &= \mathbb{E}_{\substack{x \sim \mathcal{D} \\ y \sim \pi_\theta}} \left[ \log \frac{\pi_\theta(y | x)}{\pi_r(y | x)} - \log Z(x) \right] = \mathbb{E}_{x \sim \mathcal{D}} \left[ D_{\text{KL}}(\pi_\theta(\cdot | x) | \pi_r(\cdot | x)) - \log Z(x) \right] \end{aligned}$$

where

$$\pi_r(y | x) = \frac{\pi^{\text{SFT}}(y | x) \exp \left( \frac{1}{\beta} r(x, y) \right)}{Z(x)}, \quad Z(x) = \sum_y \pi^{\text{SFT}}(y | x) \exp \left( \frac{1}{\beta} r(x, y) \right)$$

# Closed-form solution of KL-regularized RL

$$-\frac{1}{\beta} \mathcal{J}(\pi_\theta; r) = \mathbb{E}_{x \sim \mathcal{D}} \left[ D_{\text{KL}}(\pi_\theta(\cdot | x) | \pi_r(\cdot | x)) - \log Z(x) \right], \quad \pi_r(y | x) = \frac{\pi^{\text{SFT}}(y | x) \exp\left(\frac{1}{\beta} r(x, y)\right)}{Z(x)}$$

Therefore,

$$\underset{\theta}{\text{maximize}} \quad \mathcal{J}(\pi_\theta; r)$$

is equivalent to

$$\underset{\theta}{\text{minimize}} \quad \mathbb{E}_{x \sim \mathcal{D}} \left[ D_{\text{KL}}(\pi_\theta(\cdot | x) | \pi_r(\cdot | x)) - \log Z(x) \right]$$

which in turn is equivalent to

$$\underset{\theta}{\text{minimize}} \quad \mathbb{E}_{x \sim \mathcal{D}} \left[ D_{\text{KL}}(\pi_\theta(\cdot | x) | \pi_r(\cdot | x)) \right]$$

Since  $Z(x)$  not depend on  $\pi_\theta$ . Note that the normalization constant  $Z(x)$  is not tractable, but we will soon see that its value it not needed.

If we ignore the neural network parameterization, we see that the optimum policy is attained at  $\pi_\theta = \pi_r$ .

# Closed-form solution of KL-regularized RL

We have found the closed form solution mapping  $r$  to the (KL-regularized) optimal policy  $\pi_r$

$$\pi_r(y | x) = \frac{\pi^{\text{SFT}}(y | x) \exp\left(\frac{1}{\beta} r(x, y)\right)}{Z(x)}$$

Inversely, if given arbitrary policy  $\pi$ , the reward function  $r_\pi$  that makes the policy  $\pi$  optimal is

$$r_\pi(x, y) = \beta \log \frac{\pi(y | x)}{\pi^{\text{SFT}}(y | x)} + \beta \log Z(x).$$

(Generally, this is called inverse RL; Given a behavior of a rational agent, what is the reward that is being maximized?)

We have a one-to-one mapping between  $r$  and  $\pi$ .



# Direct preference optimization (DPO)

PPO-based RLHF:

1. Collect human preference data.
2. Learn  $r$  to fit the preference data.
3. Learn  $\pi$  maximizing  $r$  subject to KL penalty.

Direct preference optimization (DPO):

1. Collect human preference data.
2. Learn  $\pi$  such that  $r_\pi$  fits the preference data.

In other words, DPO uses the one-to-one parameterization between  $\pi$  and  $r$  to eliminate  $r$  and directly learn  $\pi$ .

# Direct preference optimization (DPO)

Recall that the reward model was trained via

$$\underset{\psi}{\text{minimize}} \quad \sum_{\substack{(x, y_i, y_j) \in \mathcal{D} \\ y_i > y_j}} -\log \sigma(r_\psi(x, y_i) - r_\psi(x, y_j))$$

DPO substitutes

$$r_\psi \leftarrow r_\pi(x, y) = \beta \log \frac{\pi(y | x)}{\pi^{\text{SFT}}(y | x)} + \beta \log Z(x).$$

and solves

$$\underset{\theta}{\text{minimize}} \quad \sum_{\substack{(x, y_i, y_j) \in \mathcal{D} \\ y_i > y_j}} -\log \sigma\left(\beta \log \frac{\pi_\theta(y_i | x)}{\pi^{\text{SFT}}(y_i | x)} - \beta \log \frac{\pi_\theta(y_j | x)}{\pi^{\text{SFT}}(y_j | x)}\right)$$

Note,  $Z(x)$  cancels out.

# Direct preference optimization (DPO)

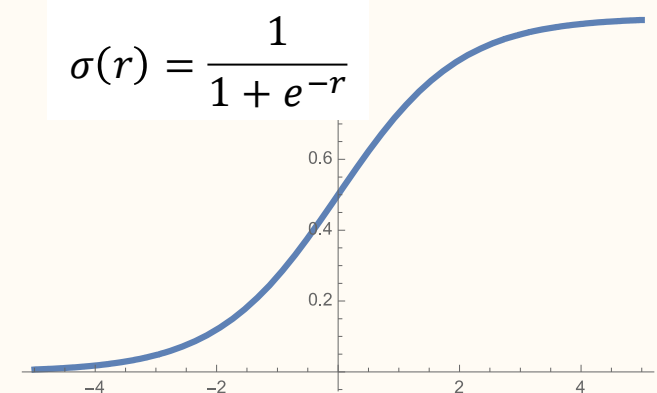
$$\underset{\theta}{\text{minimize}} \quad \sum_{\substack{(x, y_i, y_j) \in \mathcal{D} \\ y_i > y_j}} -\log \sigma \left( \beta \log \frac{\pi_{\theta}(y_i | x)}{\pi^{\text{SFT}}(y_i | x)} - \beta \log \frac{\pi_{\theta}(y_j | x)}{\pi^{\text{SFT}}(y_j | x)} \right)$$

The need to train a reward model  $r_{\psi}$  has now been removed. Also, the need for the value network  $V_{\phi}$  is also removed, since we are no longer doing the PPO optimization.

DPO effectively converts the RL problem into a supervised learning problem, and therefore DPO is so much easier to execute.

However, there is some disagreement on whether the DPO performs as well as PPO.<sup>#</sup>

# DPO loss interpretation



$$\underset{\theta}{\text{minimize}} \quad \underbrace{\sum_{\substack{(x, y_i, y_j) \in \mathcal{D} \\ y_i > y_j}} -\log \sigma \left( \beta \log \frac{\pi_{\theta}(y_i | x)}{\pi^{\text{SFT}}(y_i | x)} - \beta \log \frac{\pi_{\theta}(y_j | x)}{\pi^{\text{SFT}}(y_j | x)} \right)}_{=\mathcal{L}^{\text{DPO}}(\theta)}$$

The DPO loss itself is somewhat difficult to interpret, but its gradient provides some intuition.

$$\nabla_{\theta} \mathcal{L}^{\text{DPO}}(\theta) = -\beta \sum_{\substack{(x, y_i, y_j) \in \mathcal{D} \\ y_i > y_j}} \sigma \left( \underbrace{-(r_{\pi_{\theta}}(x, y_i) - r_{\pi_{\theta}}(x, y_j))}_{=\text{reward model error}} \right) \left( \underbrace{\nabla_{\theta} \log \pi_{\theta}(y_i | x)}_{\text{increase likelihood of } y_i} - \underbrace{\nabla_{\theta} \log \pi_{\theta}(y_j | x)}_{\text{decrease likelihood of } y_j} \right)$$

So, DPO is also doing ascent on the good completion  $y_i$  while doing descent on the bad completion  $y_j$ . The gradient is accentuated if the implicitly defined reward model  $r_{\pi_{\theta}}$  disagrees with the human annotation  $y_i > y_j$ .

# Chain-of-thought

Chain-of-thought (CoT) is a technique for LLMs to talk (think) to itself before producing an answer. Compared to immediately producing an answer, CoT greatly improves performance.

One way to induce CoT is through the prompt: "Let's think step by step."

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: The answer (arabic numerals) is

(Output) 8 **X**

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

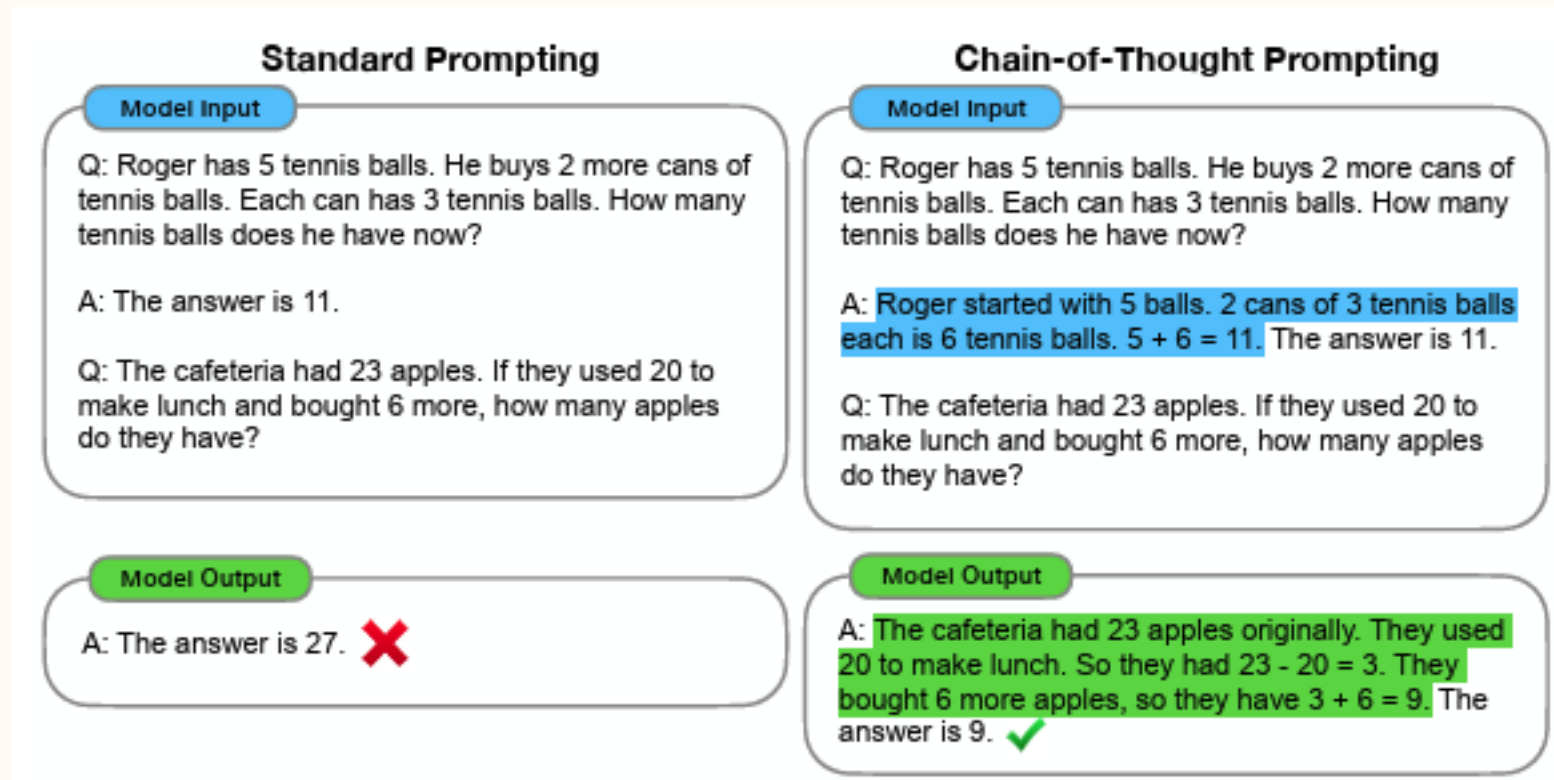
A: **Let's think step by step.**

(Output) *There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓*

# Chain-of-thought prompting

The CoT behavior can also be induced through ICL prompting.

Modern LLMs are instruction-fine-tuned to exhibit the CoT behavior.



# Co-training language models with code

Automatic program synthesis was a longstanding challenge.  
One day, LLMs emerged as a solution.

“... early investigation of GPT-3 revealed that it could generate simple programs from Python docstrings. While rudimentary, this capability was exciting because GPT-3 was not explicitly trained for code generation.”<sup>#</sup>

Code data with comments written in natural language is plentiful on the internet. Now, LLMs are explicitly trained on code together with language.

Despite the differences between code and natural language, this works surprisingly well.

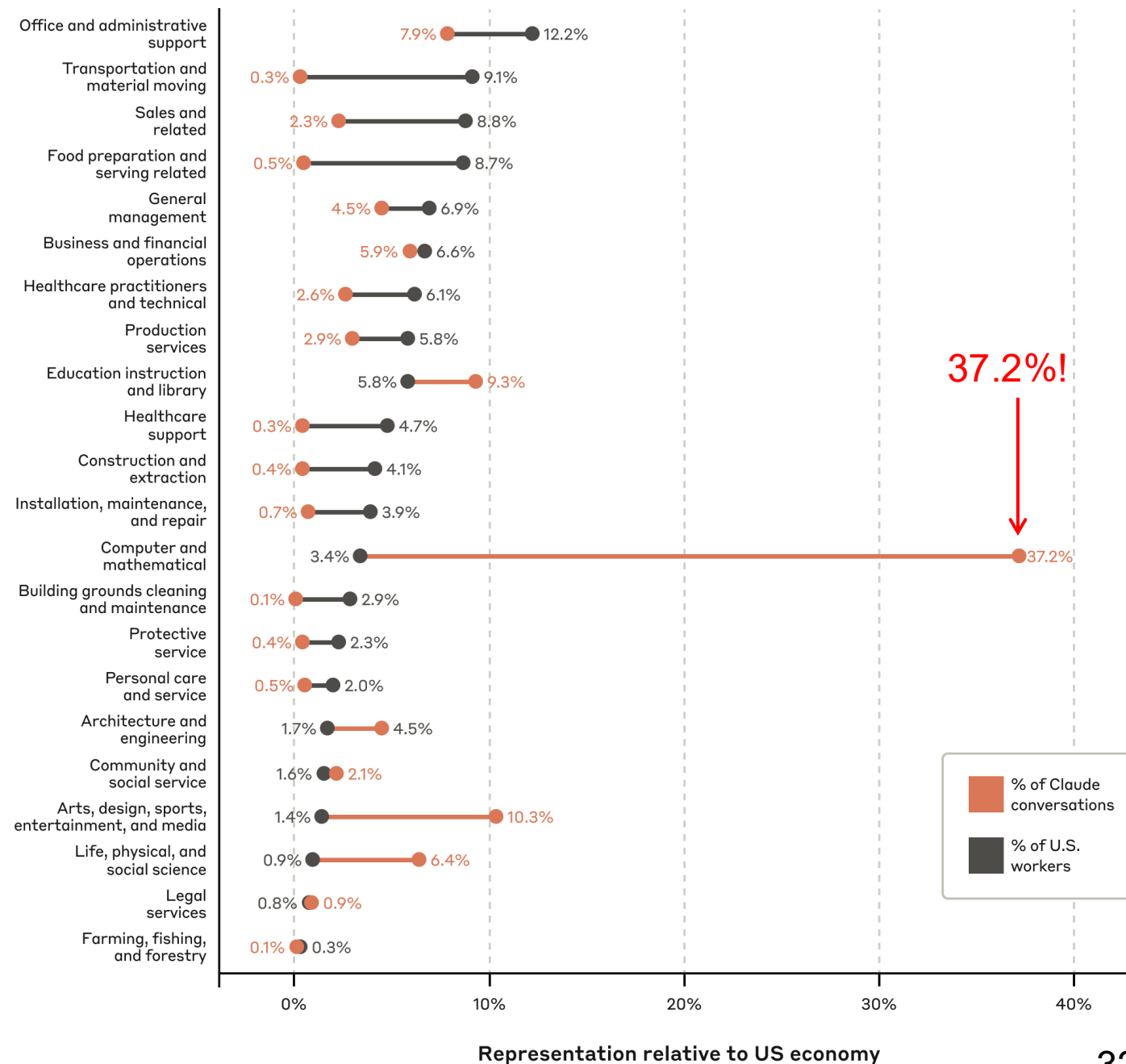
Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, CodeBERT: A pre-trained model for programming and natural languages, *ACL*, 2020.

<sup>#</sup>M. Chen, J. Tworek, ... I. Sutskever, and W. Zaremba, Evaluating large language models trained on code, *arXiv*, 2021.

# Coding is the main use of LLMs

Anthropic's report shows that 37.2% of the tokens they generate are for coding.

K. Handa, A. Tamkin, M. McCain, S. Huang, E. Durmus, S. Heck, J. Mueller, J. Hong, S. Ritchie, T. Belonax, K. K. Troy, D. Amodei, J. Kaplan, J. Clark, and D. Ganguli, Which economic tasks are performed with AI? Evidence from millions of Claude conversations, *arXiv*, Feb. 2025.





# Code training benefits logical reasoning

Interesting, coding improves natural-language (non-coding) reasoning tasks.

“Code training prior to math training improves models’ ability to solve mathematical problems ... code training improve[s] ... mathematical reasoning.”#

“we empirically show that [formal logic problems] enhances the reasoning capabilities of state-of-the-art LLMs”&

“we find a consistent results that code is a critical building block for generalization far beyond coding tasks ... outsized impact across all tasks.”%

#Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo, DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, *arXiv*, Feb. 2024.

&T. Morishita, G. Morio, A. Yamaguchi, and Y. Sogawa, Enhancing reasoning capabilities of LLMs via principled synthetic logic corpus, *NeurIPS*, 2024.

%V. Aryabumi, Y. Su, R. Ma, A. Morisot, I. Zhang, A. Locatelli, M. Fadaee, A. Üstün, and S. Hooker, To code or not to code? Exploring impact of code in pre-training, *ICLR*, 2025.

# Math problem solving with LLMs

There is a lot of research on using LLMs to solve math problems. There are 2 types:

1. Solving math problems (often at high-school or lower level) in natural language and reporting the numerical answer at the end.
2. Producing mathematical proofs (often no numerical answer) in natural language or in a formal proof assistant such as Lean.

To avoid the discussion of what a formal proof is, I will only talk about the first type.

# Math datasets

GSM-8k: Elementary and middle school level problems.

**Problem:** 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?

Math500: High school to early undergraduate level problems

**Problem:** If the domain of the function  $\log x^2$  is  $x < a$  or  $x > b$ , for some  $a$  and  $b$ , find  $a + b$ .

# Math datasets

American Invitational Mathematics Examination 2024 (AIME 2024).

**Problem:** Define  $f(x) = ||x| - \frac{1}{2}|$  and  $g(x) = ||x| - \frac{1}{4}|$ . Find the number of intersections of the graphs of

$$y = 4g(f(\sin(2\pi x))) \quad \text{and} \quad x = 4g(f(\cos(3\pi y))).$$

AIME problems are designed to have integer solutions between 0 and 999. Unlike, say the AMO or IMO, only the answer is graded.

Grading the intermediate derivations or proofs is difficult, so these benchmarks are evaluated only on the final numerical answer.

# Why solve math with AI?

- Training AI on math may improve logical reasoning in other areas.
  - This is why most humans are taught math.
- Math is useful, and AI may be able to assist us making novel mathematical discoveries.
  - E.g. Terence Tao, Javier Gomez-Serrano, and Google DeepMind are in active collaboration towards this goal.<sup>#</sup>
  - Discovering novel mathematics with AI has already been done. The next goal is to make discoveries that are significant such that they would be of interest regardless of the involvement of AI and would be publishable in a top mathematics journal.
- Mathematics does not require external real-world data (unlike other scientific disciplines).
- Among domains with objective verification mechanisms, mathematics arguably represents the pinnacle of intellectual endeavor. (Board games are just games .)

<sup>#</sup><https://mathstodon.xyz/@tao/114508029896631083>

<sup>#</sup><https://deepmind.google/discover/blog/alphaevolve-a-gemini-powered-coding-agent-for-designing-advanced-algorithms/>

# DeepSeekMath

DeepSeekMath paper's contribution:

1. Math problem data is already plentiful in commonly used pre-training dataset. Proposed a method to extract them.
2. Proposed GRPO and RL-trained LLM to solve these math problems with CoT reasoning.

This is an instance RL with Verifiable Rewards (RLVR). The reward is exact, so one cannot overfit.

Only *outcome rewards* are provided. Intermediate rewards on partial completions, called *process rewards* are not used.

# DeepSeek GRPO

while (not converged)

sample a problem  $x$

sample  $G$  responses  $y^{(1)}, \dots, y^{(G)} \sim \pi_{\theta_{\text{curr}}}(\cdot | x)$

evaluate the rewards  $\mathbf{r} = (r^{(1)}, \dots, r^{(G)})$  for  $y^{(1)}, \dots, y^{(G)}$

solve:

$$\underset{\theta_{\text{next}} \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^G \frac{1}{|T^{(i)}|} \sum_{t=0}^{T^{(i)}} (\text{term}_1 + \text{term}_2)$$

Note) length normalization

$$\text{term}_1 = \mathcal{C}_{\varepsilon_C} \left( \frac{\pi_{\theta_{\text{next}}}(y_{t+1}^{(i)} | x, y_{1:t}^{(i)})}{\pi_{\theta_{\text{curr}}}(y_{t+1}^{(i)} | x, y_{1:t}^{(i)})}, \frac{r^{(i)} - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r}) + \varepsilon_A} \right)$$

$$\text{term}_2 = -\beta \left( \frac{\pi^{\text{SFT}}(y_{s+1}^{(i)} | x^{(i)}, y_{1:s}^{(i)})}{\pi_{\theta_{\text{next}}}^{\text{RL}}(y_{s+1}^{(i)} | x^{(i)}, y_{1:s}^{(i)})} - \log \left( \frac{\pi^{\text{SFT}}(y_{s+1}^{(i)} | x^{(i)}, y_{1:s}^{(i)})}{\pi_{\theta_{\text{next}}}^{\text{RL}}(y_{s+1}^{(i)} | x^{(i)}, y_{1:s}^{(i)})} \right) - 1 \right)$$

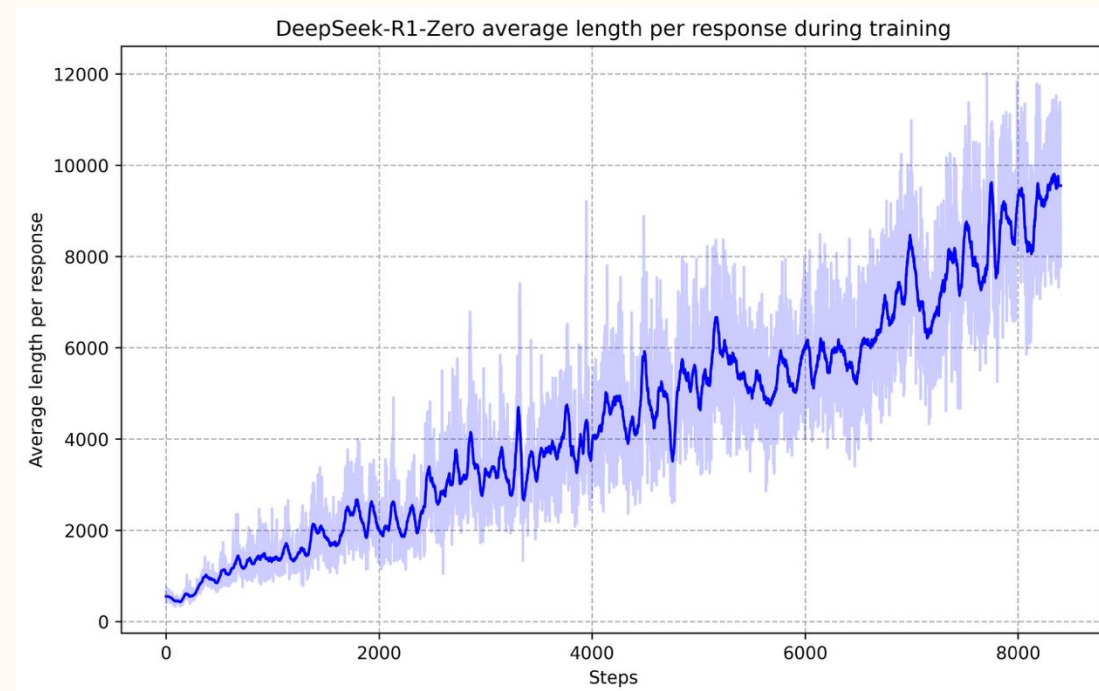
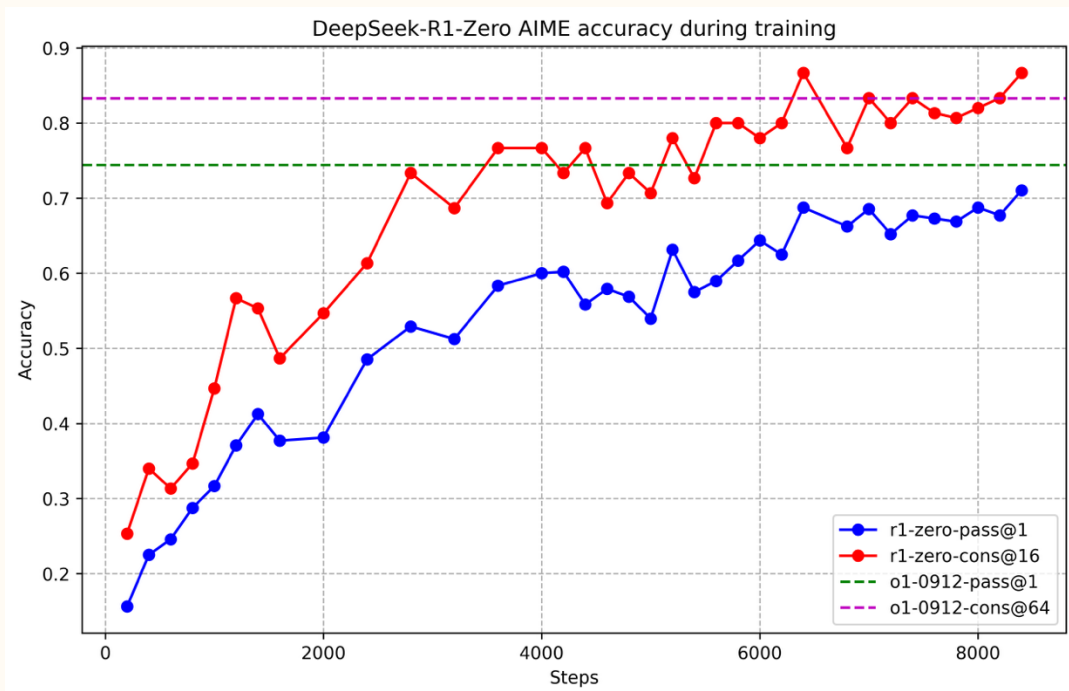
$$\theta_{\text{curr}} = \theta_{\text{next}}$$

end

# DeepSeek-R1

Scaled up the RLVR on coding, mathematics, science, logic reasoning etc.

The model learns to utilize CoT more extensively and performance improves significantly.





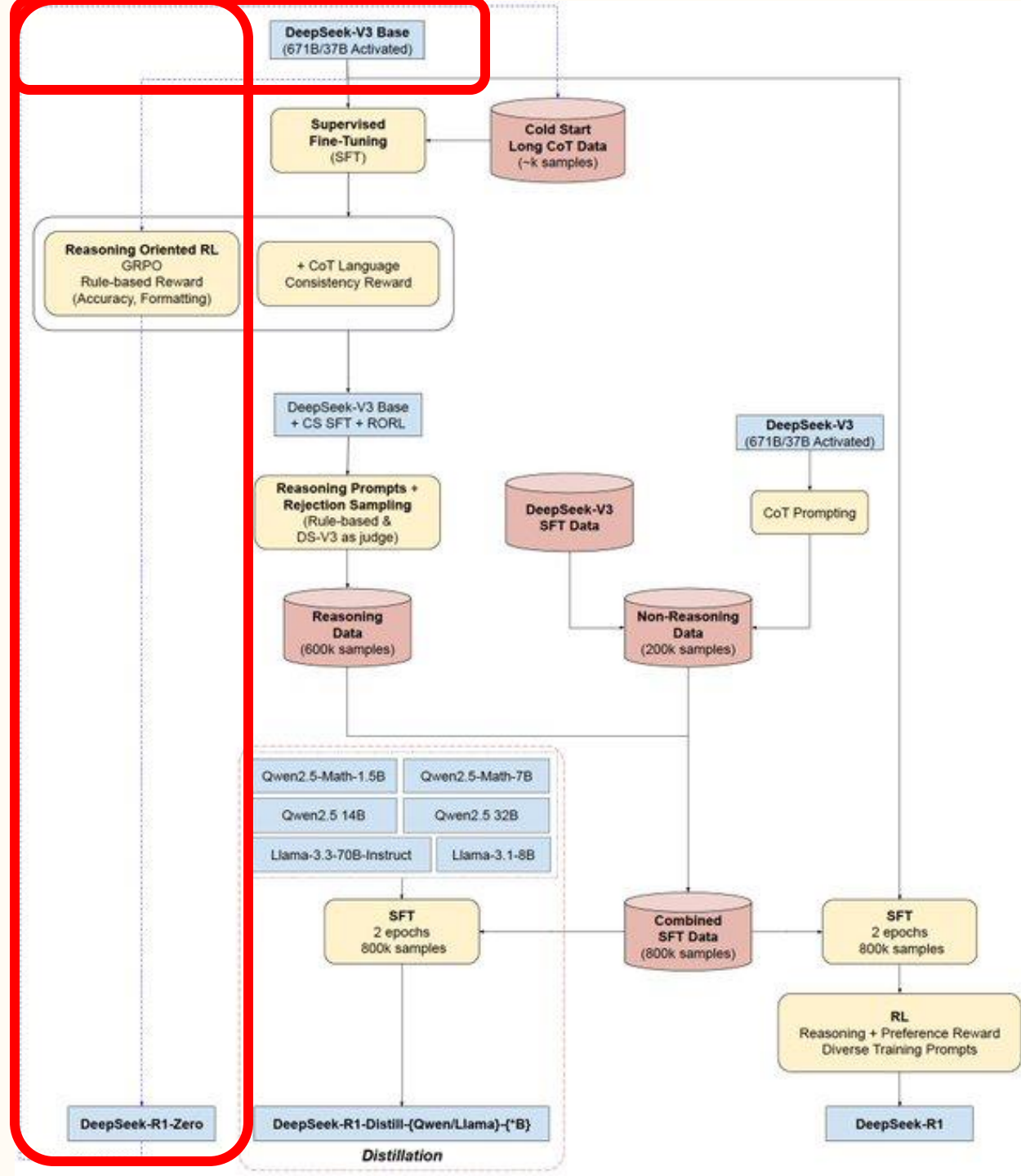
# DeepSeek-R1 detailed training pipeline

DeepSeek-V3 Base model is a pre- and post-trained LLM with several interesting low-level architectural innovations. RLHF is used, but no RLVR yet.

RLVR applied to DeepSeek-V3 Base yields DeepSeek-R1-**Zero**. This model learns to use longer CoT and performance improves, but the CoT suffers from poor readability.

DeepSeek-R1 has many additional steps. We highlight a few.

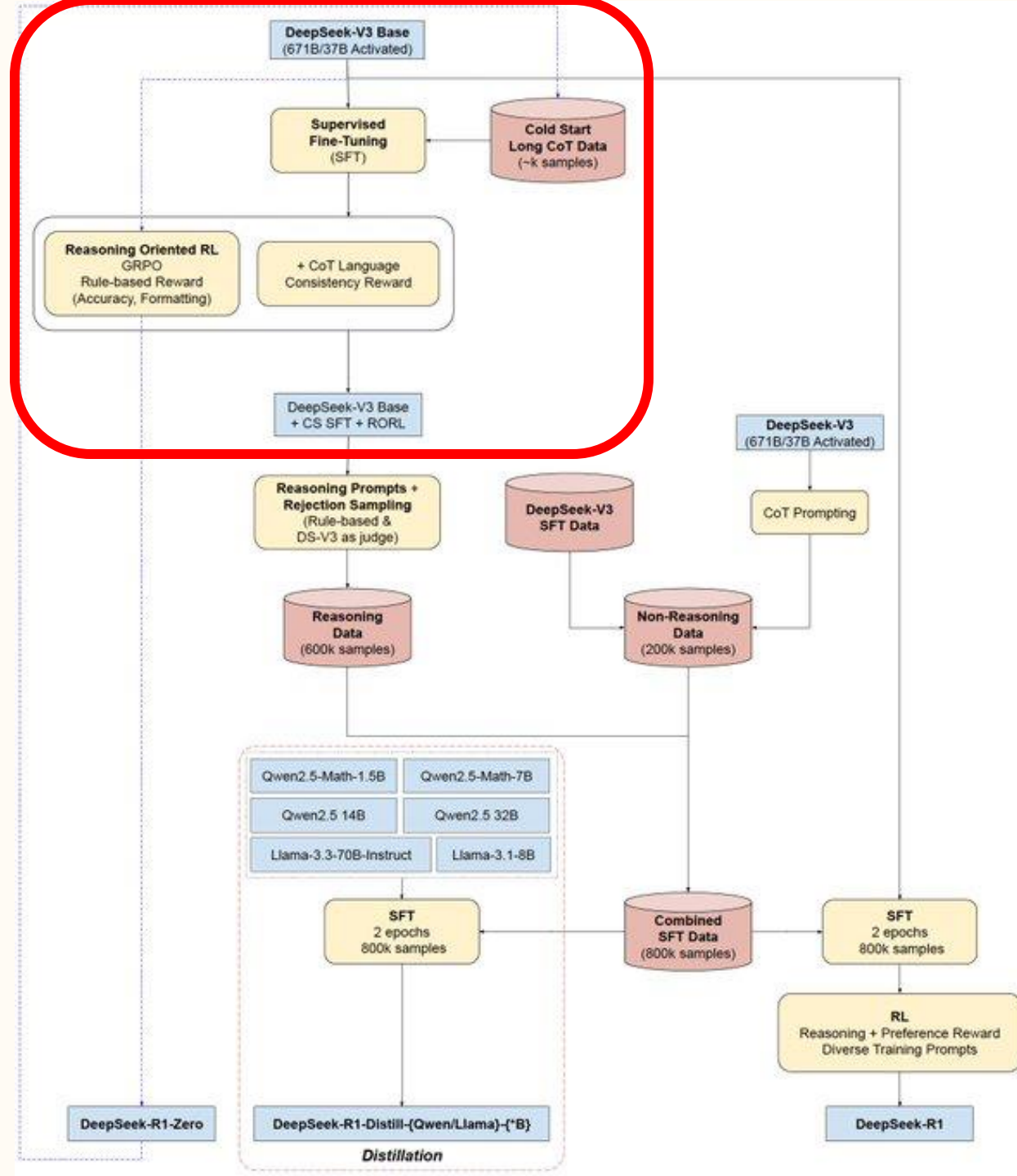
DeepSeek-AI, DeepSeek-V3 technical report, *arXiv*, Dec. 2024.  
Image Source: <https://x.com/SirrahChan/status/1881540279783887036>



# DeepSeek-R1 detailed training pipeline

Collect some good CoT examples and use them to train the base model with supervised fine-tuning (SFT, next-token prediction). Use SFT'd model to warm start the RLVR process. (The authors call this a “cold start”.)

To prevent CoT language from alternating between English and Chinese, impose a language consistency loss in RL.

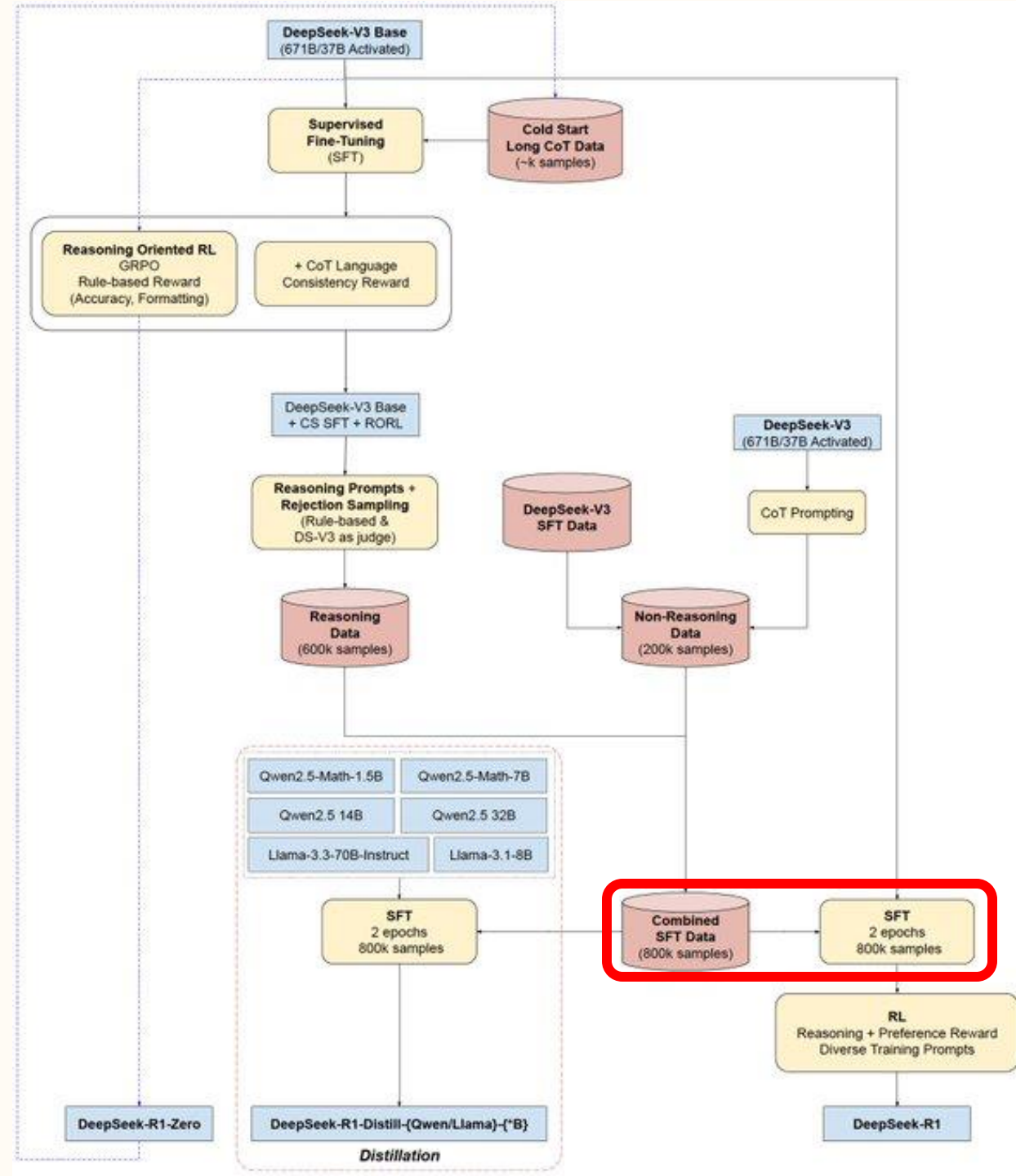


# DeepSeek-R1 detailed training pipeline

We want the model to be general-purpose, not just good for code and math.

Build a dataset with good CoT traces for verifiable and non-verifiable (e.g., creative writing) tasks.

SFT-train the DeepSeek-V3 Base model on this dataset.

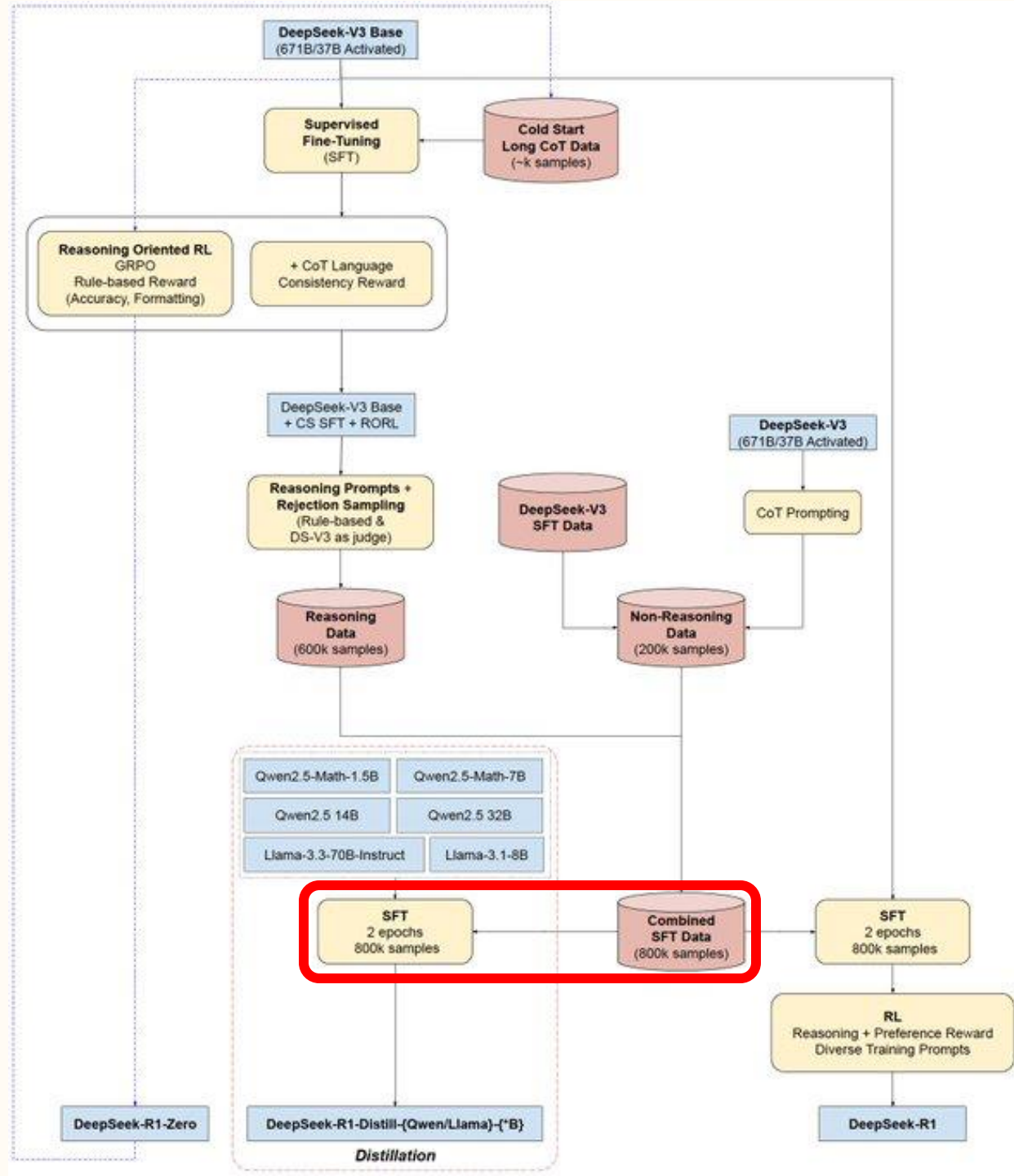


# DeepSeek-R1 detailed training pipeline

DeepSeek-R1 and DeepSeek-V3 Base are 671B parameter models. (They are MoE so the number of activated parameters is smaller.)

Distillation takes the final SFT dataset and trains smaller pre-trained Qwen and Llama models.

The distilled models perform much better than directly training these models with RLVR.

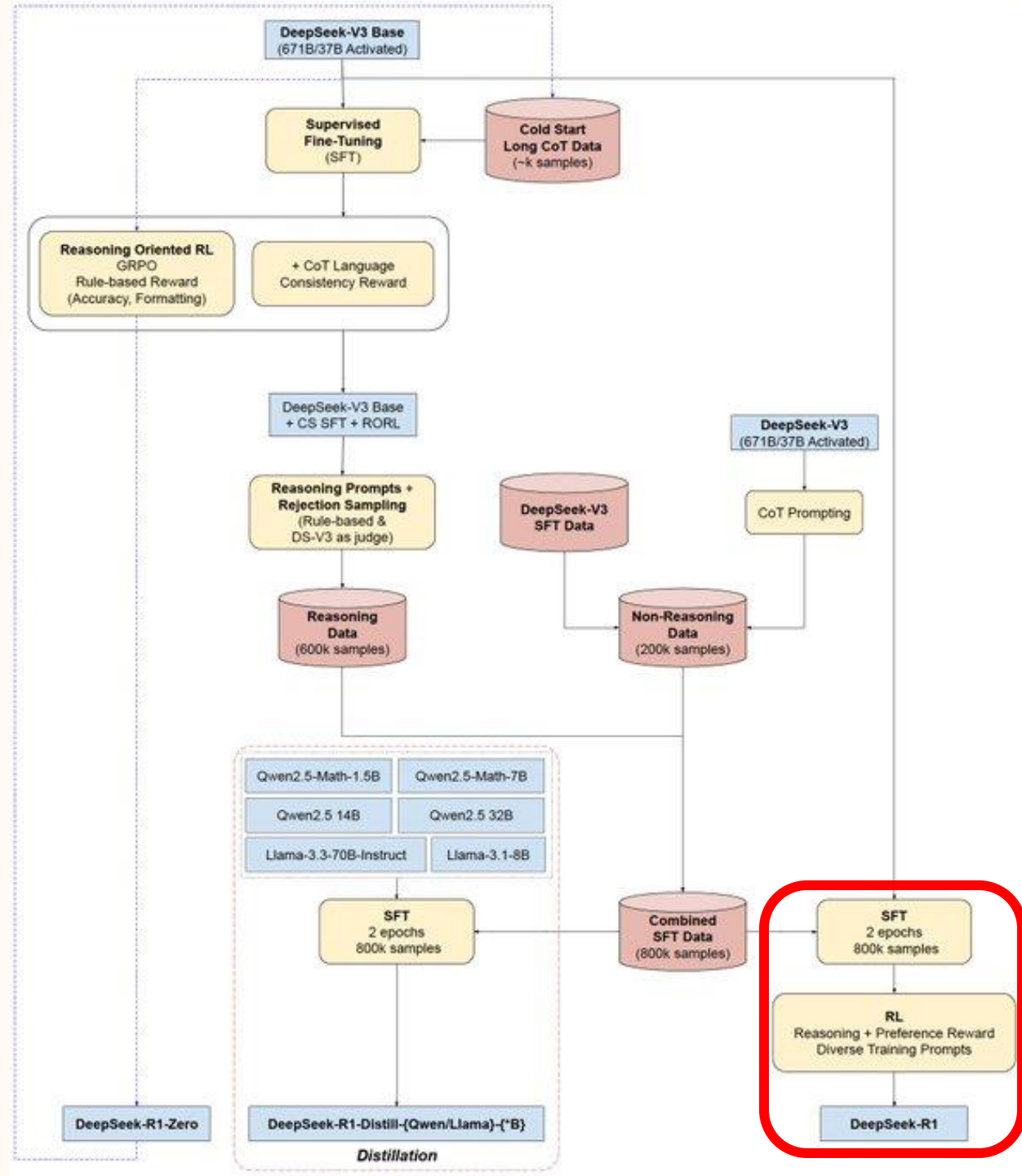




# DeepSeek-R1 detailed training pipeline

Final model DeepSeek-R1 is produced by further RL training after SFT.

The final RL training combines RLVR with RLHF, which uses GRPO with a reward model trained on human feedback. (No PPO or DPO.)



# System prompt and special token for DeepSeek-R1

---

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:

---

The special tokens `<think>` and `</think>` are introduced to enclosed the CoT reasoning traces that the outcome reward function should ignore.

“We intentionally limit our constraints to this structural format, avoiding any content-specific biases—such as mandating reflective reasoning or promoting particular problem-solving strategies—to ensure that we can accurately observe the model’s natural progression during the RL process.”

# Emergent abilities of LLMs

*Emergence* refers to the phenomenon where complex and often unexpected behaviors or capabilities arise from a system as it scales, particularly when these behaviors were not explicitly programmed or evident in smaller versions of the system.

The term is frequently misused in unscientific contexts, often due to its association with a sense of mysticism.

Many of the capabilities exhibited by LLMs are emergent.

- Nice because we acquire a capability without explicitly curated data or specialized architectural design.
- Bad because the scale at which a capability emerges (if it emerges at all) is unpredictable.

# Emergence of non-linear reasoning through RLVR

“One of the most remarkable aspects of this self-evolution is the emergence of sophisticated behaviors as the test-time computation increases. Behaviors such as reflection—where the model revisits and reevaluates its previous steps—and the exploration of alternative approaches to problem-solving arise spontaneously. These behaviors are not explicitly programmed but instead emerge as a result of the model’s interaction with the reinforcement learning environment.”



# Why did RLVR suddenly work?

In hindsight, RLVR is a fairly obvious approach, and other researchers have tried it earlier.#

Earlier attempts failed because non-linear reasoning capabilities did not emerge and improvements from RL saturated.

Why didn't these capabilities emerge in earlier models? A post-mortem analysis?

#L. Trung, X. Zhang, Z. Jie, P. Sun, X. Jin, and H. Li, ReFT: Reasoning with reinforced fine-tuning, *ACL*, 2024. (*arXiv*, Jan. 2024.)  
<https://x.com/rosstaylor90/status/1886625126222852208>



Ross Taylor ✓  
@rosstaylor90



No one is saying RL didn't work for reasoning. The argument is about internal reasoning emergence, not absolute performance boosts with RL.

Quite the opposite in fact - we had PPO on Llama 2 base models 2 years ago with verifiable rewards and had 90%+ on GSM8k. We already knew that RL worked. Hell we even did this with the bronze-age Galactica model and it also worked (see my ICML talk 3 years ago).

What didn't "work" was that we didn't see emergence of longer traces with backtracking, checking, error-correction and other branching-like behaviour.

So when people say "the base model wasn't strong enough", the contention is that there wasn't enough knowledge in the base model, subsequent RL compute and context window length for the model to develop long CoT, internal reasoning on its own.

That's the argument, not whether "RL worked". It always did, but the magic seems to have emerged relatively recently.

Last edited 12:57 PM · Feb 4, 2025 · **133.7K** Views

# RLVR works with key cognitive behaviors

Qwen and DeepSeek base models exhibit certain key cognitive behaviors for reasoning: verification (systematic error-checking), backtracking (abandoning failing approaches), subgoal setting (decomposing problems into manageable steps), backward chaining (reasoning from desired outcomes to initial inputs).

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

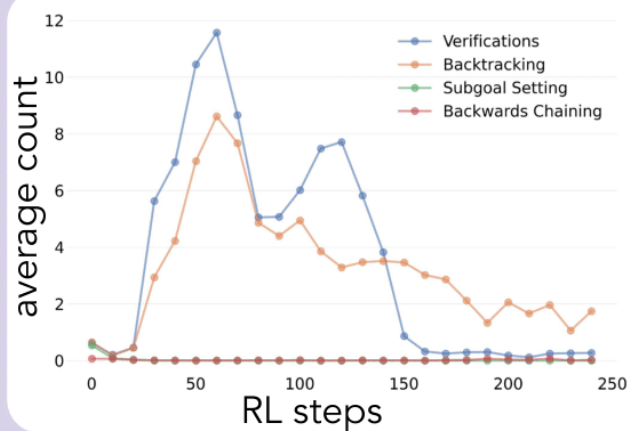
"Let's try a different approach, what if we ..."

## Backward Chaining

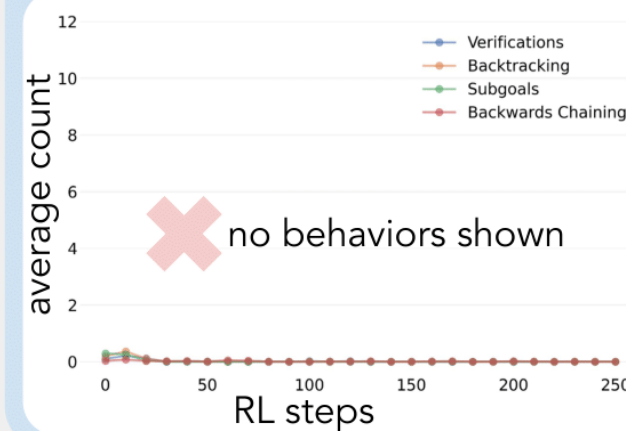
"Working backwards, 24 is 8 times 3"



Qwen




Llama




# RLVR works with key cognitive behaviors

RLVR only works for models with these cognitive behaviors. The Llama base model does not naturally exhibit these capabilities.

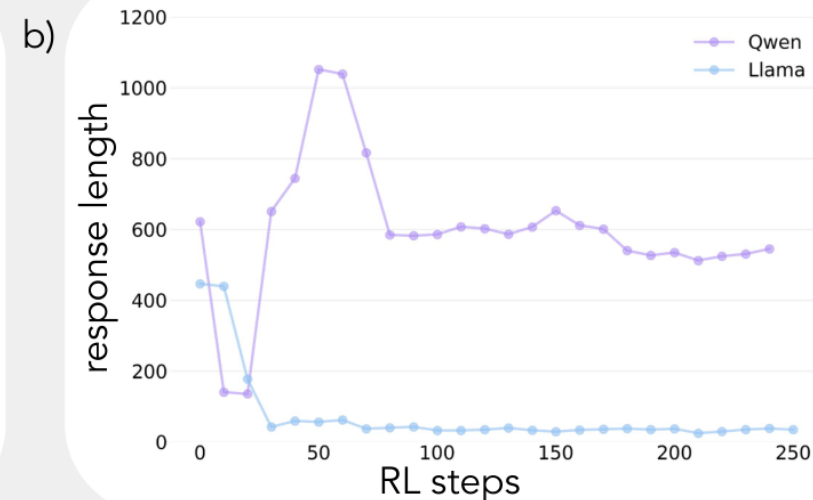
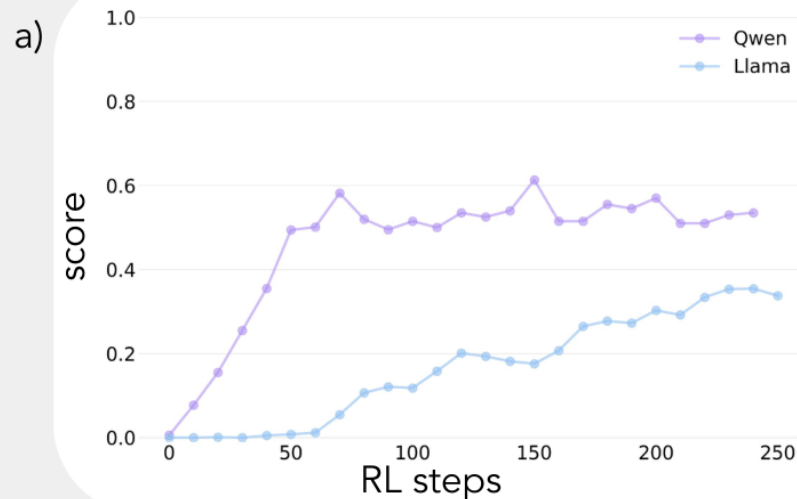
A tale of two models: [Qwen 2.5 3B](#) and [Llama 3.2 3B](#)



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.

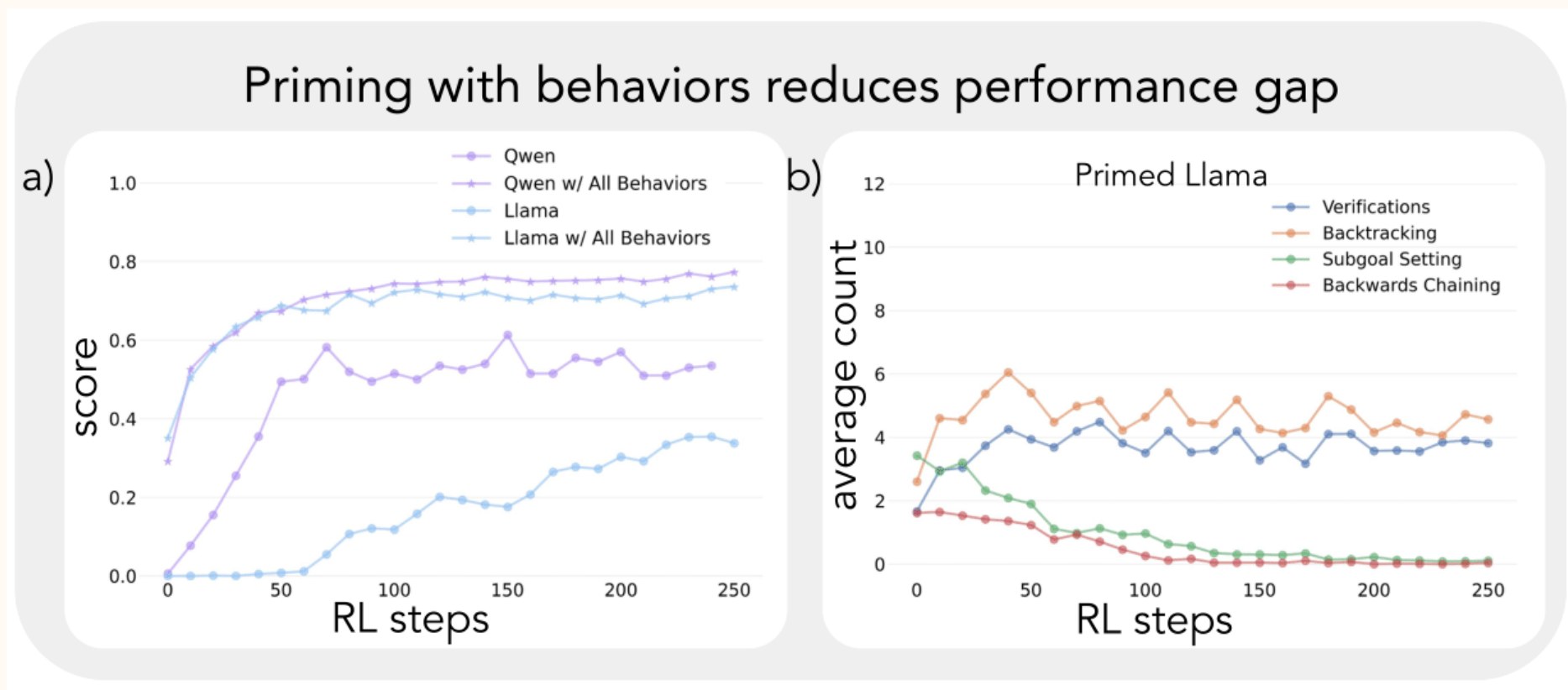


84 is the difference between 108 and 34.  
<answer>  $(84 - 34) / 108$  </answer>



# RLVR works with key cognitive behaviors

If we teach (prime) Llama these cognitive behaviors with SFT, then RLVR starts to work.

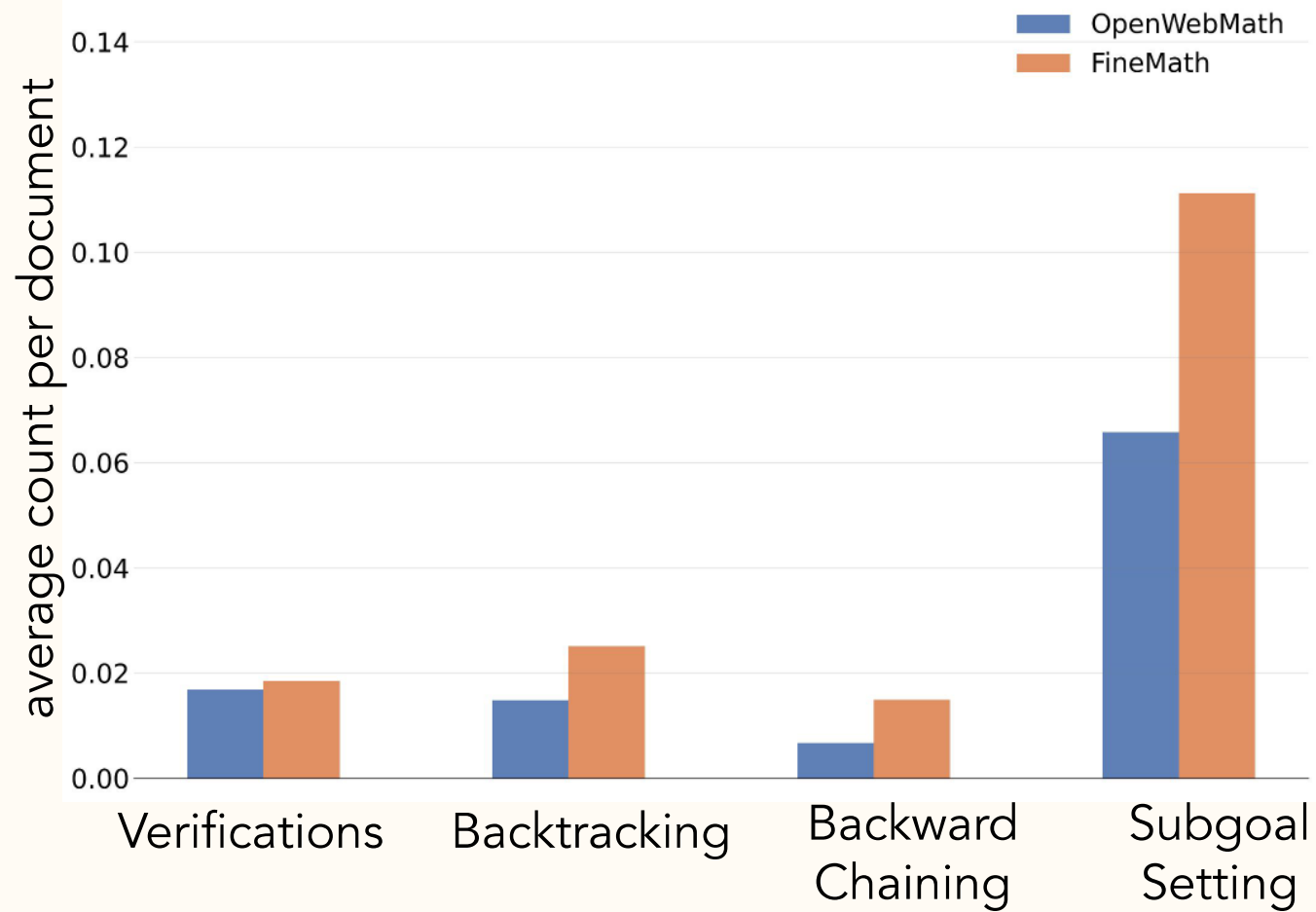


# Datasets with key cognitive behaviors

OpenWebMath<sup>#</sup> and FineMath<sup>%</sup> are math problem-solving datasets with these key cognitive behaviors.

Perhaps Llama was not on these datasets. (No public evidence on the training data.)

When Llama is further trained on OpenWebMath, RLVR starts work.



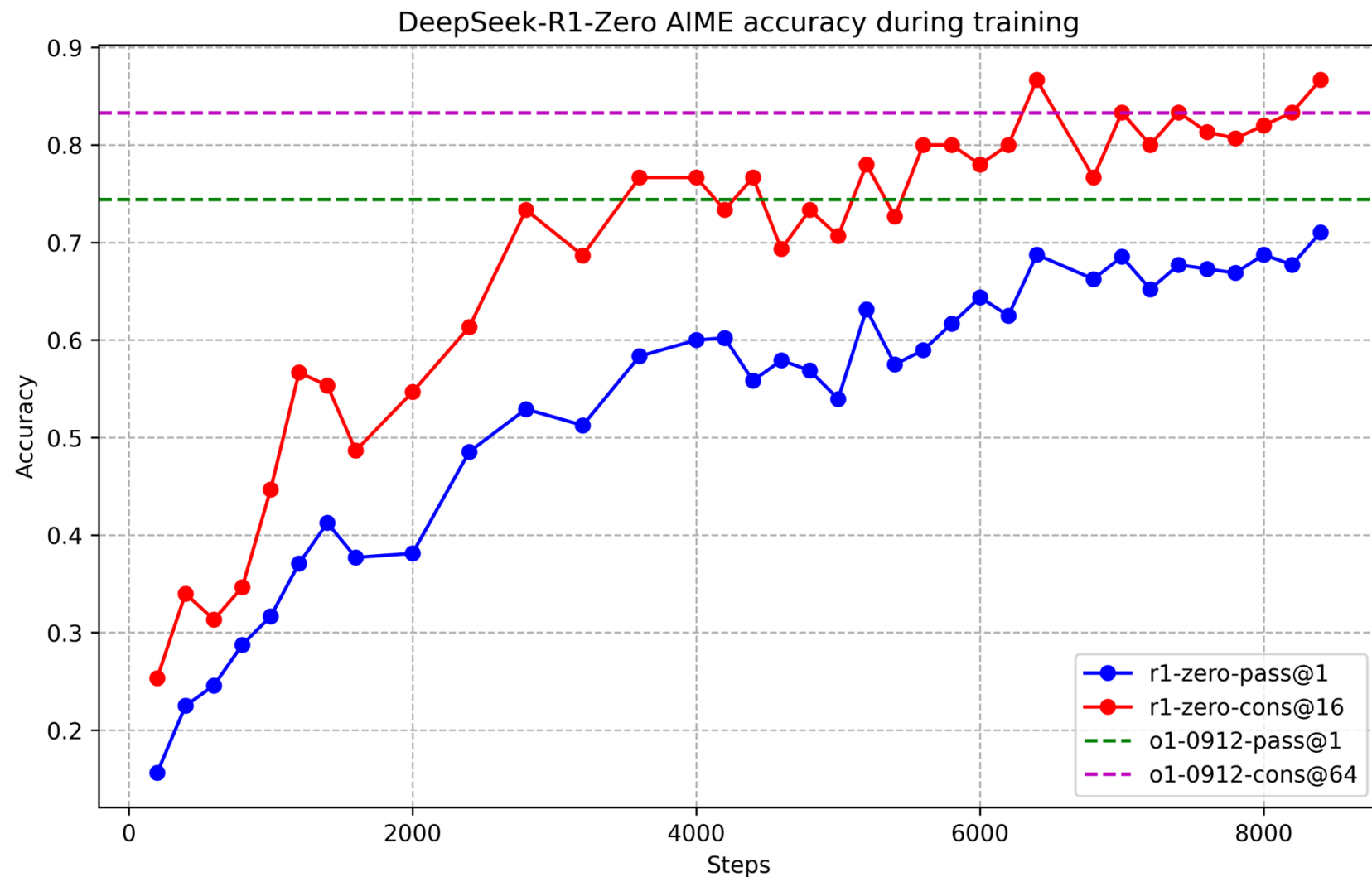
<sup>#</sup>K. Paster, M. Dos Santos, Z. Azerbayev, and J. Ba, OpenWebMath: An open dataset of high-quality mathematical web, *ICLR*, 2024.

<sup>%</sup>L. Ben Allal, A. Lozhkov, ... T. Wolf, SmolLM2: When Smol goes big — Data-centric training of a small language model, *arXiv*, Feb. 2025.

K. Gandhi, A. Chakravarthy, A. Singh, N. Lile, and N. D. Goodman, Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STaRs, *arXiv*, Mar. 2025.

# Majority vote

Majority vote, self-consistency<sup>#</sup>, or consensus is a simple technique for improving the accuracy of reasoning models. Simply generate  $N$  reasoning paths and report the most frequent answer.



DeepSeek-R1 reasoning also improves with majority vote.  
(cons@64 means consensus with 64 generations.)

<sup>#</sup>X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, Self-consistency improves chain of thought reasoning in language models, *ICLR*, 2023.

DeepSeek-AI, DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, *arXiv*, Jan. 2025.



# DeepSeek-R1 and test-time scaling

Test-time scaling refers to using more compute at test-time (during inference) to improve the quality or accuracy of the output.

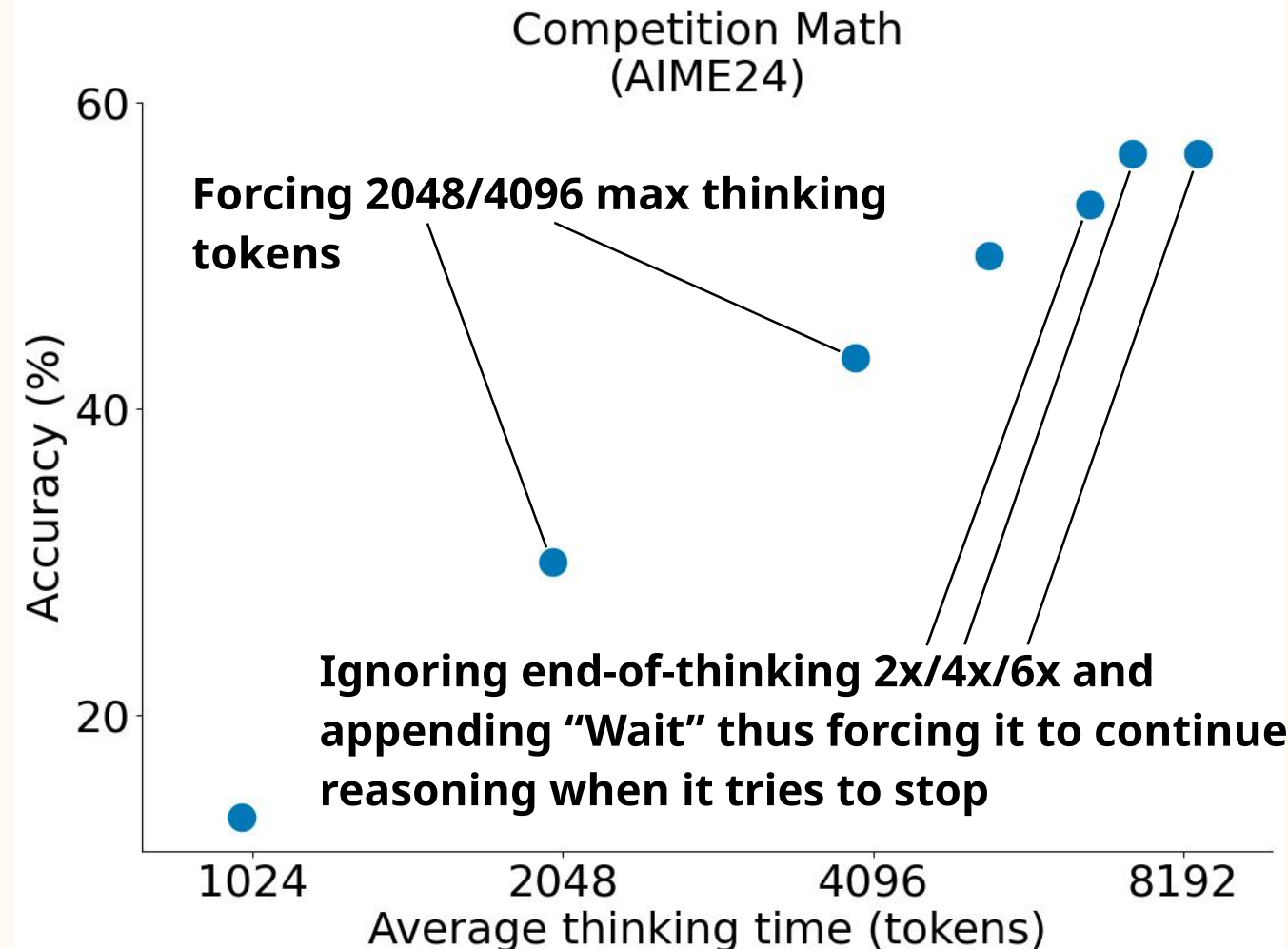
For LLMs, DeepSeek-R1 shows:

- Majority vote remains a valid approach to test-time scaling.
  - Multiple generations incurs compute cost.
- RLVR allows the model to use the CoT for test-time scaling.
  - Generating the longer CoT incurs compute cost.
  - Can we control the CoT length?

# Test-time scaling with budget forcing

Given a CoT reasoning model (not necessarily RLVR trained), we can control the CoT length using a budget forcing technique.

- To terminate the CoT early, append “Final Answer:” at the end. The model will produce an answer based on the current CoT.
- To continue the CoT that has concluded, delete the answer and append “Wait” at the end. The model will continue the reasoning.





# Unsuccessful test-time scaling results

Interestingly, the DeepSeek-R1 authors report some negative results for test-time scaling.

Process reward models (PRM) do not work well. PRMs are trained models evaluating correctness or usefulness of intermediate steps. PRMs are hard to train (requires significant human annotation), and their use leads to over-optimization and reward hacking.

MCTS also does not work well. Compared to AlphaGo, there are too many possible actions (tokens) per step, so the tree search quickly blows up in size.

MCTS has been successfully used by the DeepSeek team for when interacting with formal mathematical proof assistants,<sup>%</sup> but successfully using MCTS-like search for pure language reasoning remains an open problem.

DeepSeek-AI, DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, *arXiv*, Jan. 2025.

<sup>%</sup>H. Xin, Z. Z. Ren, ... C. Ruan, DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and Monte-Carlo tree search, IC LR, 2025.

# GRPO has some problems

Recall the GRPO algorithm, restated for comparison.

```
while (not converged)
```

$$\hat{A}^{(i)} = \frac{r^{(i)} - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r}) + \varepsilon_A}$$

```
  solve:
```

$$\underset{\theta_{\text{next}} \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^G \frac{1}{|T^{(i)}|} \sum_{t=0}^{T^{(i)}} \left( \mathcal{C}_{\varepsilon_C} \left( \frac{\pi_{\theta_{\text{next}}}(y_{t+1}^{(i)} | x, y_{1:t}^{(i)})}{\pi_{\theta_{\text{curr}}}(y_{t+1}^{(i)} | x, y_{1:t}^{(i)})}, \hat{A}^{(i)} \right) + \text{KL}_t^{(i)} \right)$$

$$\theta_{\text{curr}} = \theta_{\text{next}}$$

```
end
```

Note the **length normalization**.

# DAPO

DAPO changes the **length normalization** mechanism.

(DAPO also proposed some other changes, but we won't cover them.)

while (not converged)

$$\hat{A}^{(i)} = \frac{r^{(i)} - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r}) + \varepsilon_A}$$

solve:

$$\underset{\theta_{\text{next}} \in \mathbb{R}^p}{\text{maximize}} \quad \frac{1}{\sum_{i=1}^G |T^{(i)}|} \sum_{i=1}^G \sum_{t=0}^{T^{(i)}} \left( \mathcal{C}_{\varepsilon_C} \left( \frac{\pi_{\theta_{\text{next}}} (y_{t+1}^{(i)} | x, y_{1:t}^{(i)})}{\pi_{\theta_{\text{curr}}} (y_{t+1}^{(i)} | x, y_{1:t}^{(i)})}, \hat{A}^{(i)} \right) + \text{KL}_t^{(i)} \right)$$

$$\theta_{\text{curr}} = \theta_{\text{next}}$$

end

# Dr. GRPO

Dr. GRPO (GRPO Done Right) removes the length normalization altogether, and also removes the advantage normalization.

while (not converged)

$$\hat{A}^{(i)} = r^{(i)} - \text{mean}(\mathbf{r})$$

solve:

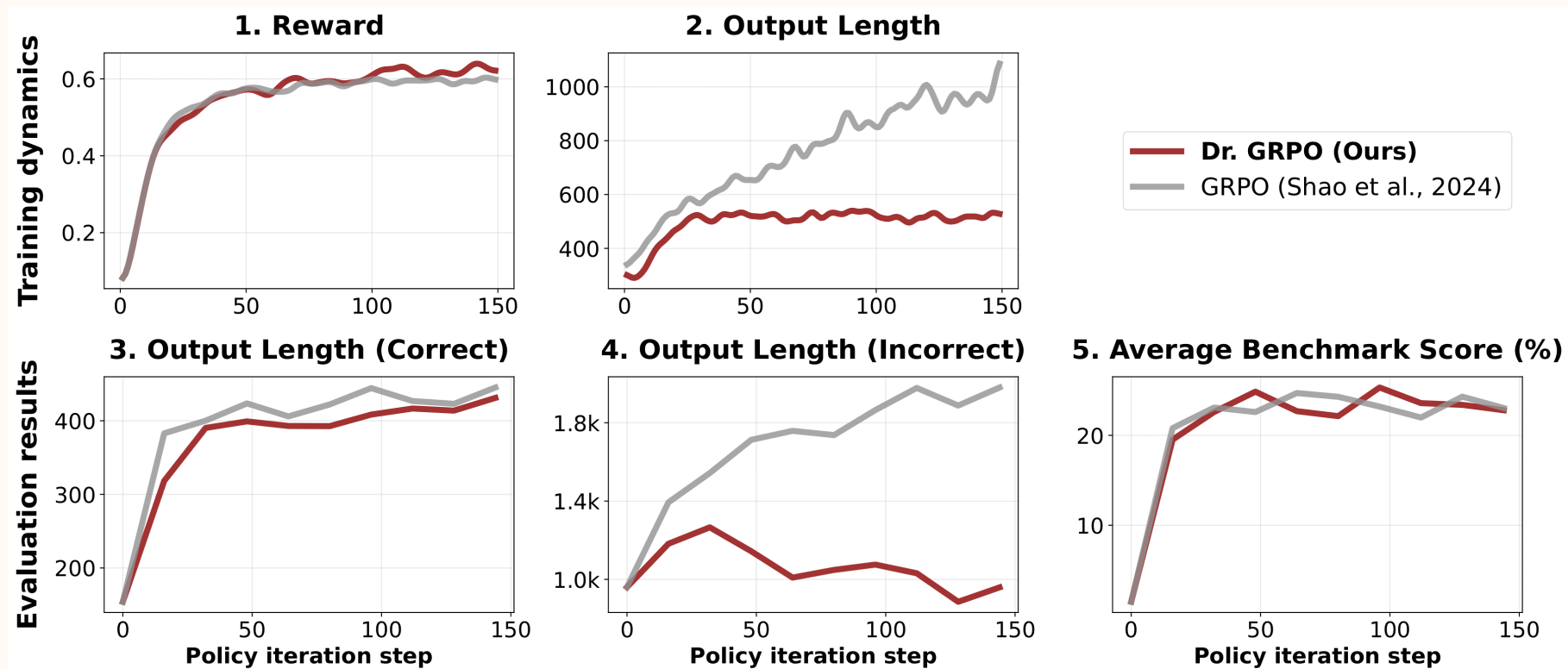
$$\underset{\theta_{\text{next}} \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^G \sum_{t=0}^{T^{(i)}} \left( \mathcal{C}_{\varepsilon_C} \left( \frac{\pi_{\theta_{\text{next}}}(y_{t+1}^{(i)} | x, y_{1:t}^{(i)})}{\pi_{\theta_{\text{curr}}}(y_{t+1}^{(i)} | x, y_{1:t}^{(i)})}, \hat{A}^{(i)} \right) + \text{KL}_t^{(i)} \right)$$

$$\theta_{\text{curr}} = \theta_{\text{next}}$$

end

# Length normalization bias $\frac{1}{|T^{(i)}|} \sum_{t=0}^{T^{(i)}} \mathcal{C}_{\varepsilon_C} \left( \frac{\pi_{\theta_{\text{next}}}(y_{t+1}^{(i)} | x, y_{1:t}^{(i)})}{\pi_{\theta_{\text{curr}}}(y_{t+1}^{(i)} | x, y_{1:t}^{(i)})}, \hat{A}^{(i)} \right)$

With length normalization, for shorter responses, correct answers are strongly preferred.



However, longer responses, correct or incorrect, do not receive much reward or penalty. So, if you think you'll get things wrong, you might as well make the response long to receive less penalty. This is what caused DeepSeek-R1's CoT to grow longer and longer.

# Question-level difficulty bias

For the GRPO advantage estimate, questions with lower standard deviations (problems that are too easy or too hard) are given higher weights during policy updates.

$$\hat{A}^{(i)} = \frac{r^{(i)} - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r}) + \varepsilon_A}$$

It's not clear if this bias causes significant problems, but Dr. GRPO removes the standard-deviation normalization to remove this bias.

$$\hat{A}^{(i)} = r^{(i)} - \text{mean}(\mathbf{r})$$

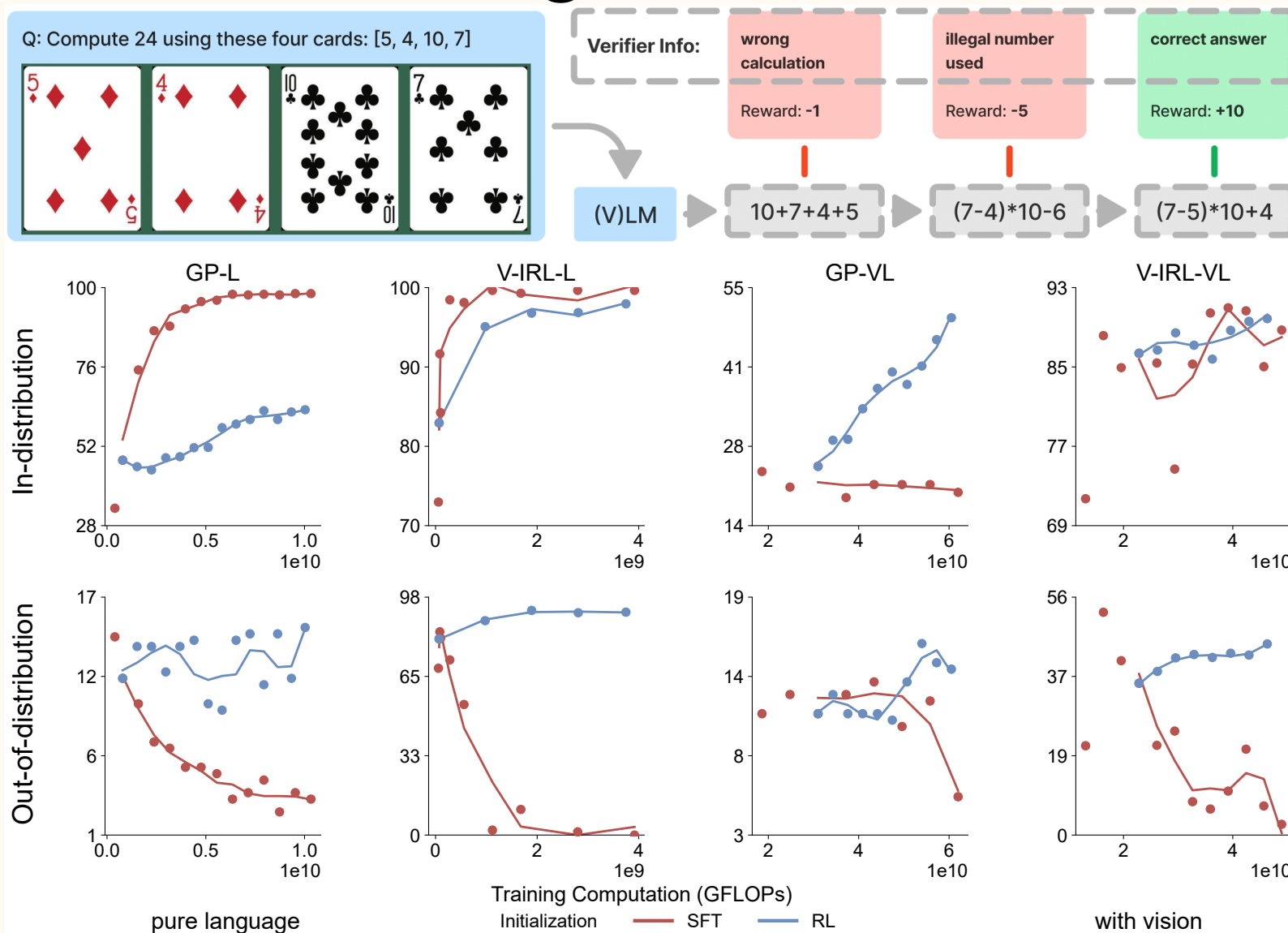
One can show that Dr. GRPO's advantage estimates corresponds to unbiased policy gradient estimates:

$$\nabla \mathcal{J}(\theta) \propto \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{i=1}^G \sum_{t=0}^{T^{(i)}} \nabla_\theta \log \pi_\theta(y_{t+1}^{(i)} | x, y_{1:t}^{(i)}) (r^{(i)} - \text{mean}(\mathbf{r})) \right]$$

# SFT memorizes and RLVR generalizes

The authors define simple toy tasks and train on them with SFT and RLVR.

Results show that RLVR leads to better generalization.



T. Chu, Y. Zhai, J. Yang, S. Tong, S. Xie, D. Schuurmans, Q. V. Le, S. Levine, and Y. Ma, SFT memorizes, RL generalizes: A comparative study of foundation model post-training, *ICML*, 2025.

# Why does RL generalize well?

My hypothesis:

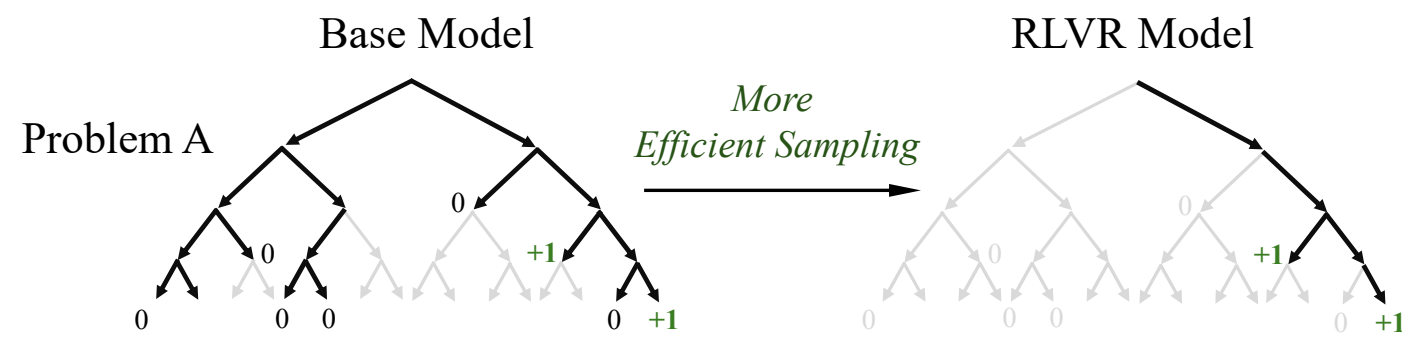
SFT lacks explicit negative signals. When a behavior is absent from the training data, the model gradually infers its undesirability through omission. This indirect signal is weak and inefficient.

By comparison, RL provides explicit negative feedback. Actions that lead to lower rewards are penalized. Although there is no credit assignment and all actions leading to a poor outcome are downweighted without fine-grained attribution, the model nonetheless receives direct signals about which behaviors to avoid.

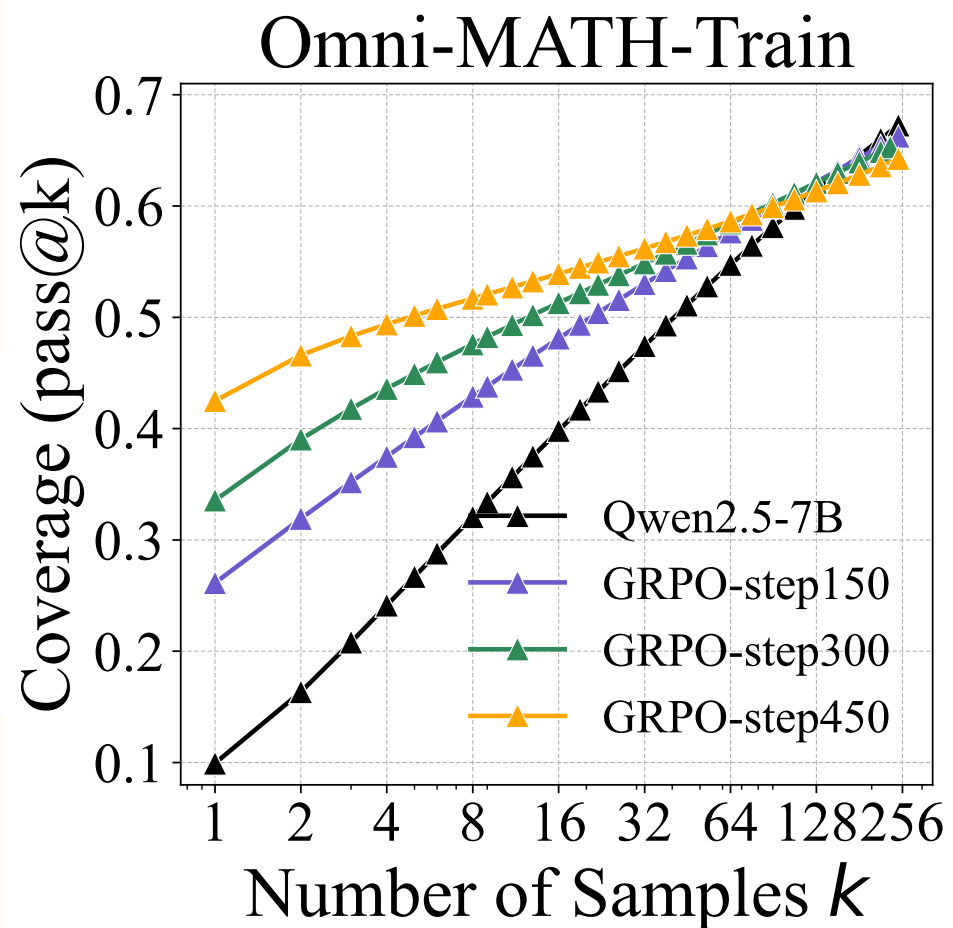


# Does RLVR learn new reasoning? No?

There is some evidence that RL does not teach new reasoning behaviors; it reinforces (makes more likely) the correct reasoning pattern that the model already knows from pre-training.

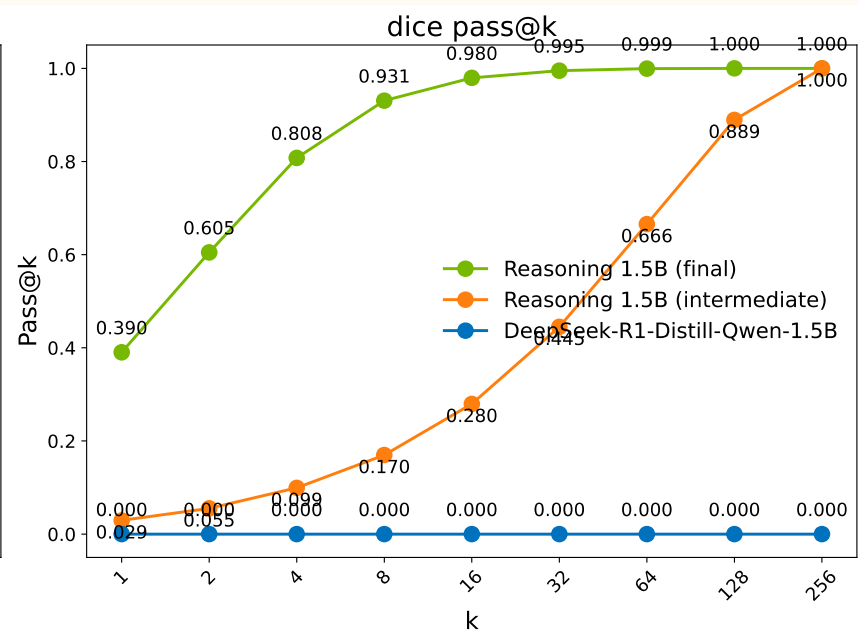
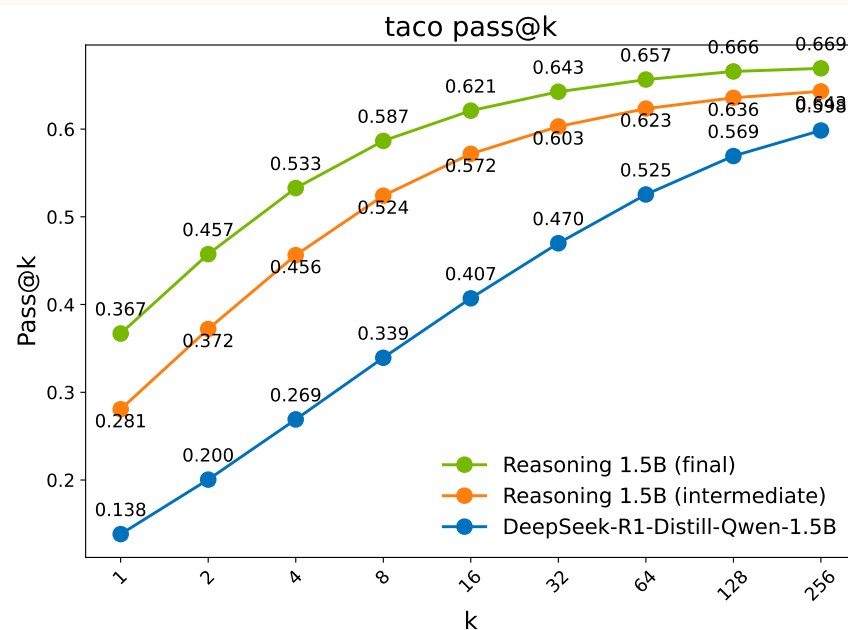
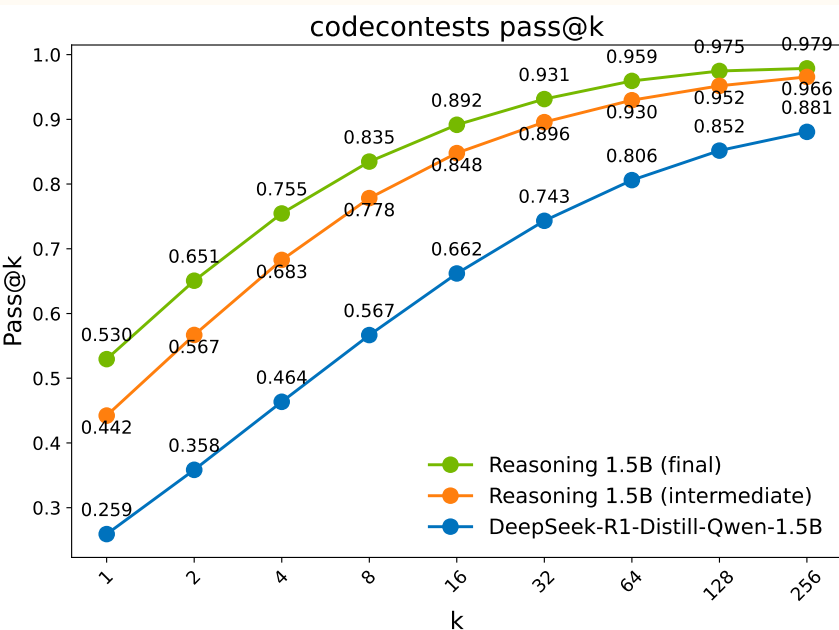


Evidence: If we evaluate the model with pass@k (answer only needs to be correct one out of k times), RLVR actually worsens the performance.



# Does RLVR learn new reasoning? Yes?

However, a follow-up paper argues that with their Prolonged RL (ProRL) technique, they can prevent “entropy collapse” and have RL learn novel reasoning strategies that are inaccessible to base models.



# Entropy collapse

A key challenge in prolonged policy optimization is *entropy collapse*, a phenomenon where the model's output distribution loses diversity early in training, resulting in sharply reduced entropy. When this happens, the policy prematurely commits to a narrow set of outputs, severely limiting exploration. Without sufficient exploration, RL stagnates.

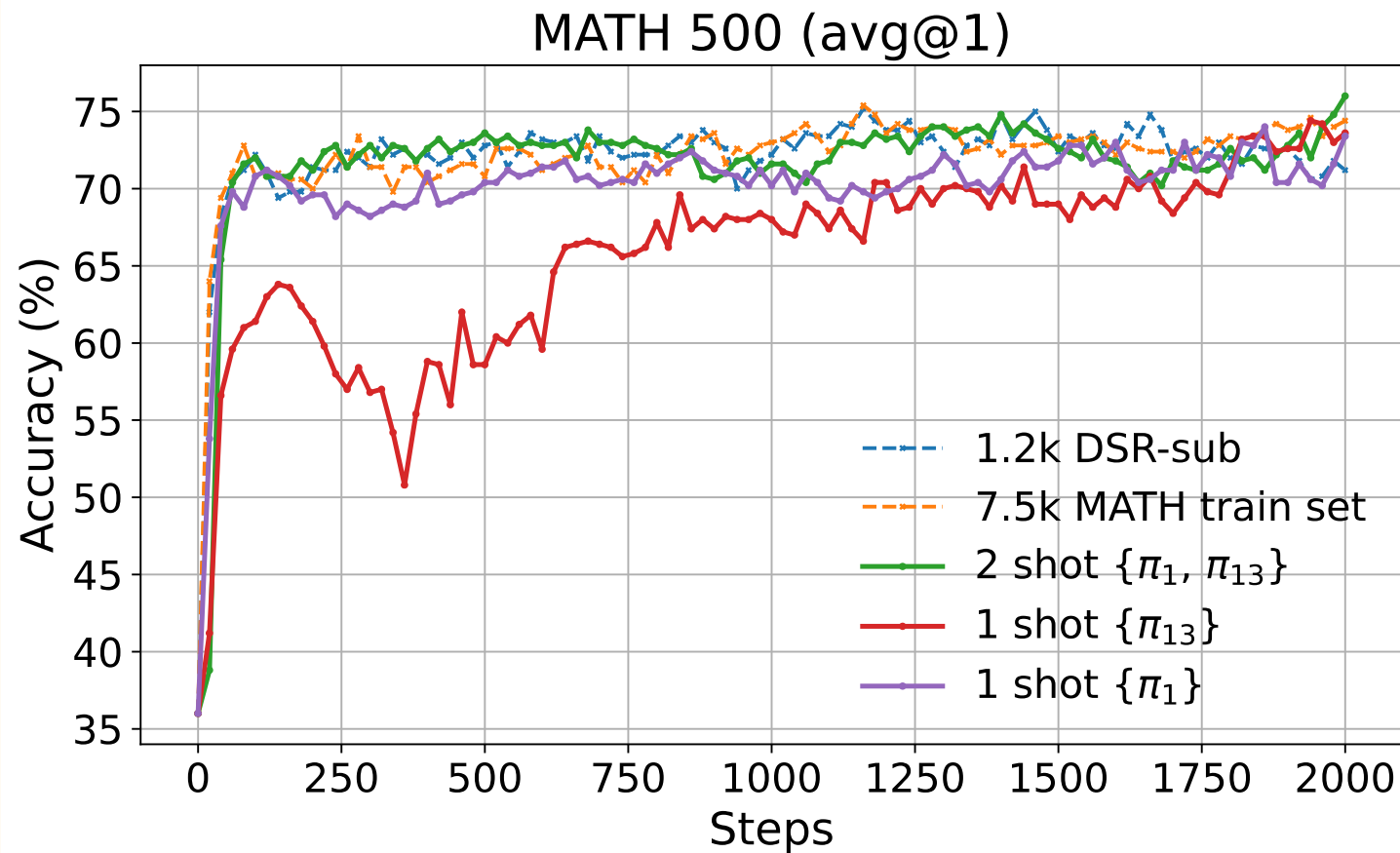
The authors propose ProRL to remedy entropy collapse. ProRL incorporates components from the DAPO training loss and periodically resets the KL penalty. (Details in the paper.)

Takeaways: (i) naïve RL with GRPO stagnates and may not be able to learn new reasoning behavior, (ii) there are ways to effectively perform prolonged RL training without stagnation, and (iii) LLMs can learn new reasoning behaviors in prolonged RL training.

# Does RLVR learn new reasoning? No?

It turns out, RLVR provides a benefit with *one* training example. RLVR works with any single problem, but some problems work better.

Hypothesis: RLVR reinforces the correct reasoning pattern that the model already knows from pre-training. A single problem generates multiple reasoning (solution) paths, and RLVR reinforces the correct reasoning steps.



The one training example:

The pressure  $P$  exerted by wind on a sail varies jointly as the area  $A$  of the sail and the cube of the wind's velocity  $V$ . When the velocity is 8 miles per hour, the pressure on a sail of 2 square feet is 4 pounds. Find the wind velocity when the pressure on 4 square feet of sail is 32 pounds. Let's think step by step and output the final answer within `\boxed{\}`.

# Fog of war

In a newly emerging and rapidly evolving field, research findings often noisy. This is because researchers are still navigating through various pitfalls, and the best practices and know-hows have yet to be established.

Therefore, it is important to approach these papers with a healthy dose of skepticism.

The following blogpost reports that some recent RL-LLM findings may be spurious.

N. Chandak, S. Goel, and A. Prabhu, Incorrect baseline evaluations call into question recent LLM-RL claims, May 2025.

<https://safe-lip-9a8.notion.site/Incorrect-Baseline-Evaluations-Call-into-Question-Recent-LLM-RL-Claims-2012f1fbf0ee8094ab8ded1953c15a37>

# Reasoning without RL, with expert iter.

RLVR and reasoning is an exciting new development (and is the focus of this course), but RL using policy-gradient-type methods is not the only way to induce reasoning capabilities in LLMs.

One non-RL<sup>#</sup> approach is expert iteration:

for  $n = 0, 1, \dots$

- Given a model  $M_n$ , get an enhanced model  $M_n^+$ .
  - Enhancement technique 1: Use test-time scaling approaches like majority voting or MCTS based on the model  $M_n$ .
  - Enhancement technique 2: Use outcome verification on behavior of  $M_n$ .
- Train next model  $M_{n+1}$  by fitting  $M_n^+$ .

<sup>#</sup>Some may argue that expert iteration is also an RL method, although I personally think it feels more closer to supervised learning (SL).

# Self-Taught Reasoner (STaR)

We can use few-shot ICL prompts to make a model CoT reasoning without RL. We can enhance the CoT reasoning with expert iteration.

Idea #1) Rationale generation: Generate CoT traces and keep ones with a correct answer. (Outcome verification.)

Q: What can be used to carry a small dog?

- (a) swimming pool
- (b) basket
- (c) dog show
- (d) backyard
- (e) own home

A: The answer must be something that can be used to carry a small dog. Baskets are designed to hold things. Therefore, the answer is basket (b).

# Self-Taught Reasoner (STaR)

Idea #2) Rationalization: For problems that the model got wrong, tell the answer to the model and have it generate an explanation justifying the answer.

Q: Where do you put your grapes just before checking out?

Answer Choices:

- (a) mouth
- (b) grocery cart (CORRECT)
- (c) super market
- (d) fruit basket
- (e) fruit market

A: The answer should be the place where grocery items are placed before checking out. Of the above choices, grocery cart makes the most sense for holding grocery items. Therefore, the answer is grocery cart (b).

If the model returns the correct answer at the end, remove the **hint**, and use the data as if the model generated the CoT on its own.



# Self-Taught Reasoner (STaR)

Idea #3) Iterate this process.

---

## Algorithm 1 STaR

---

**Input:**  $M$ : a pretrained LLM; dataset  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^D$  (No CoT, only outcome data  $y_i$ .)

- 1:  $M_0 \leftarrow M$  # Copy the original model
- 2: **for**  $n = 0, \dots, N - 1$  **do**
- 3:    $(\hat{r}_i, \hat{y}_i) \leftarrow M_n(x_i) \quad \forall i \in [1, D]$  # Perform rationale generation
- 4:    $(\hat{r}_i^{\text{rat}}, \hat{y}_i^{\text{rat}}) \leftarrow M_n(\text{add\_hint}(x_i, y_i)) \quad \forall i \in [1, D]$  # Perform rationalization
- 5:    $\mathcal{D}_n \leftarrow \{(x_i, \hat{r}_i, y_i) \mid i \in [1, D] \text{ and } \hat{y}_i = y_i\}$  # Filter rationales using ground truth answers
- 6:    $\mathcal{D}_n^{\text{rat}} \leftarrow \{(x_i, \hat{r}_i^{\text{rat}}, y_i) \mid i \in [1, D] \text{ and } \hat{y}_i \neq y_i \text{ and } \hat{y}_i^{\text{rat}} = y_i\}$  # Filter rationalized rationales
- 7:    $M_{n+1} \leftarrow \text{train}(M, \mathcal{D}_n \cup \mathcal{D}_n^{\text{rat}})$  # Finetune the original model on correct solutions
- 8: **end for**

---

In our expert iteration notation,  $\mathcal{D}_n \cup \mathcal{D}_n^{\text{rat}}$  is generated by  $M_n^+$ , and the dataset gradually improves since (i) more problems get solved and (ii) more problems get solved without rationalization.

# Self-Taught Reasoner (STaR)

Without using policy-gradient-type updates, CoT reasoning is learned.

	GSM8K Test Accuracy (%)	Train Data Used (%)
Few-shot Direct GPT-J	3.0	~0
Few-shot CoT GPT-J	3.1	~0
GPT-J Direct Finetuned	5.8	100
STaR without rationalization	10.1	25.0
STaR with rationalization	<b>10.7</b>	28.7

# Recursive self-improvement

Consider tasks with an unambiguous notion of difficulty such as adding two integers each with  $n$  or fewer digits. So, for each problem  $x_i$ , write  $\text{difficulty}(x_i)$  to denote the difficulty.

Key observation) If  $M_n$  is trained on  $x_i$  with  $\text{difficulty}(x_i) \leq n$ , then  $M_n$  can solve  $x_i$  with  $\text{difficulty}(x_i) = n + 1$  with a reasonable accuracy and this accuracy can be boosted with majority voting.

# Recursive self-improvement

Train  $M_n$  on data  $\mathcal{D} = \{(x_i, y_i)\}$  such that  $\text{difficulty}(x_i) \leq n$ .

for  $n = 0, 1, \dots$

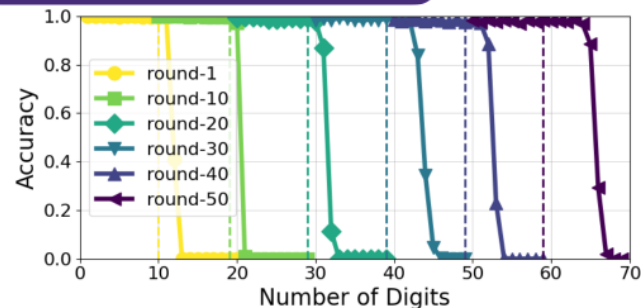
- majority voting, generate data  $\mathcal{D}_{n+1} = \{(x_i, y_i)\}$  with  $\text{difficulty}(x_i) = n + 1$ .
- Set  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{n+1}$  and train  $M_{n+1}$  on  $\mathcal{D}$ .

Notably, only data on easy ( $\text{difficulty}(x_i) \leq n$ ) instances are used. Model learns to solve harder instances without labels on hard instances. (Easy-to-hard generalization.)

In our expert iteration notation,  $M_n^+$  is  $M_n$  with majority voting. The test-time-scaling-enhanced  $M_n^+$  is good at solving problems of difficulty  $n + 1$ , even though  $M_n$  is not very accurate on such instances.

# Recursive self-improvement

## Forward Addition



10-digit (round 1)

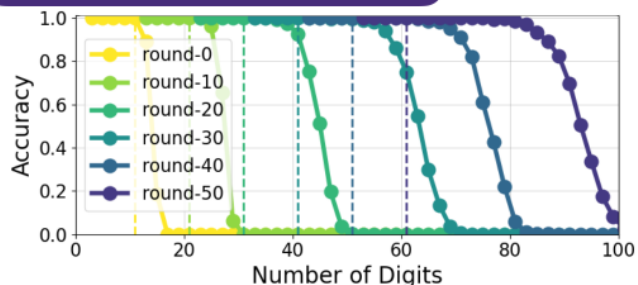
Digits ↑

19-digit (round 10)

1687477129+1095477427=  
2782954556

3642507227842806162+491464396  
5279623131=557151193122429293

## String Copying



Length 10 (round 1)

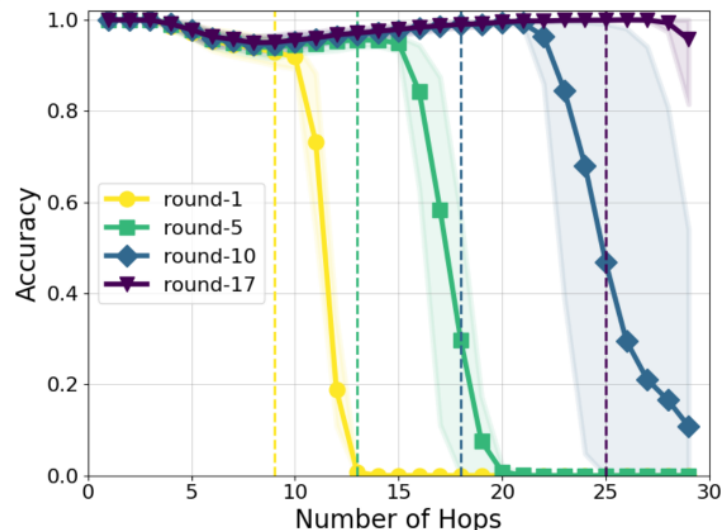
Length ↑

Length 19 (round 10)

1095477427=  
1095477427

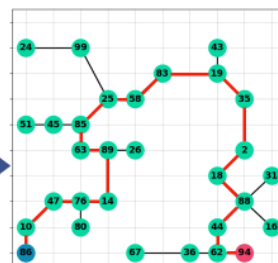
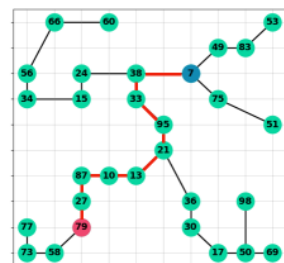
4914643965279623131=  
4914643965279623131

## Maze-Solving



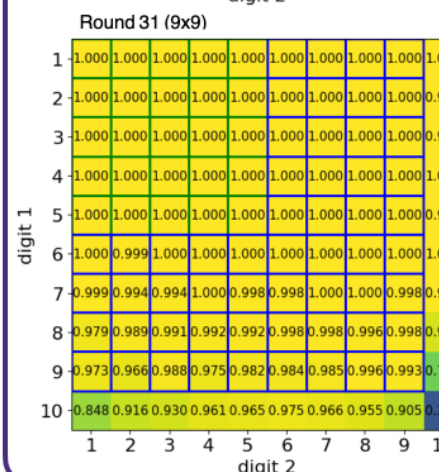
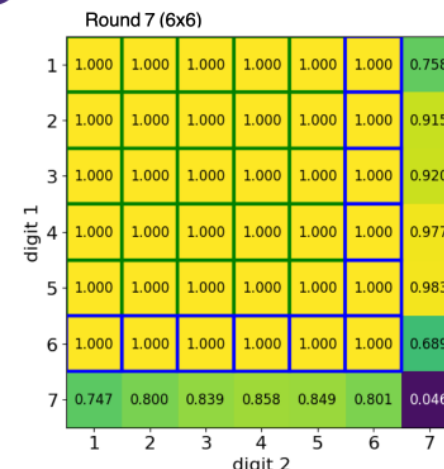
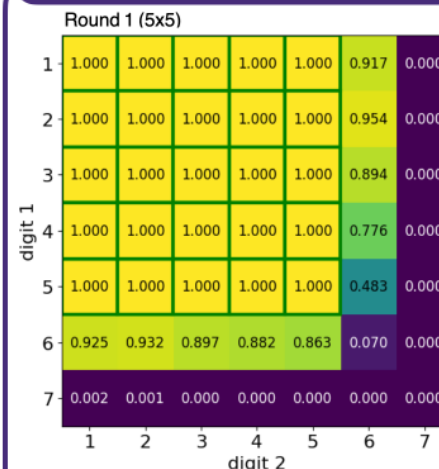
9 Hops (round 1)

18 Hops (round 10)



Hops ↑

## Multiplication



5 x 5 (round 1)

69173\*19434=  
691730+0467433(6384833)+00487841(63  
236281)+000885111(632158921)+000048  
7841=6321967161

6 x 6 (round 7)

678514\*157328=  
6785140+00839702(67690212)+00231119  
2(678223213)+0008267421(6780599551)+  
00002571380(67807477890)+0000080072  
33=678072875243

Digits ↑

# Conclusion

We are in the summer of RL and AI.

These are exciting times.