PRIME: Process Reinforcement through Implicit Rewards

https://github.com/PRIME-RL/PRIME

Ganqu Cui

2025.02.22

Pre-training will End?

Ilya Sutskever at NeurIPS 2024

- Go beyond imitation

Pre-training as we know it will end

Compute is growing:

- Better hardware
- Better algorithms
- Larger clusters

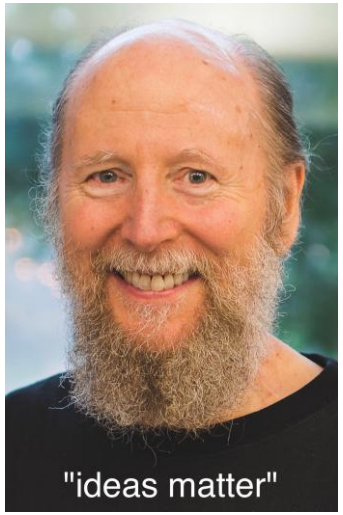
Data is not growing:

- We have but one internet
- **The fossil fuel of AI**

Why Reinforcement Learning

The next Scaling Law?

One thing that should be learned from the bitter lesson is the great power of general purpose methods, of methods that continue to **scale with increased computation** even as the available computation becomes very great. The two methods that seem to scale arbitrarily in this way are ***search* and *learning***.



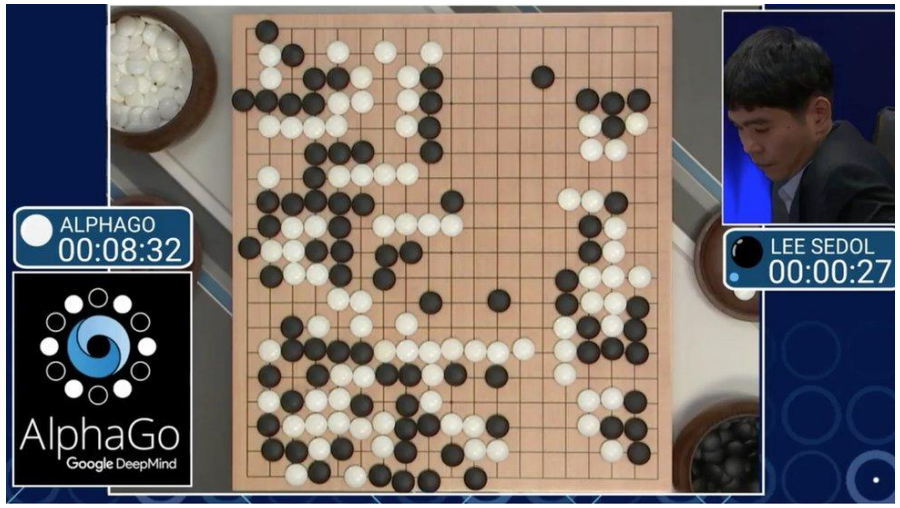
Richard Sutton
The Bitter Lesson

Reinforcement learning

Pretraining and finetuning

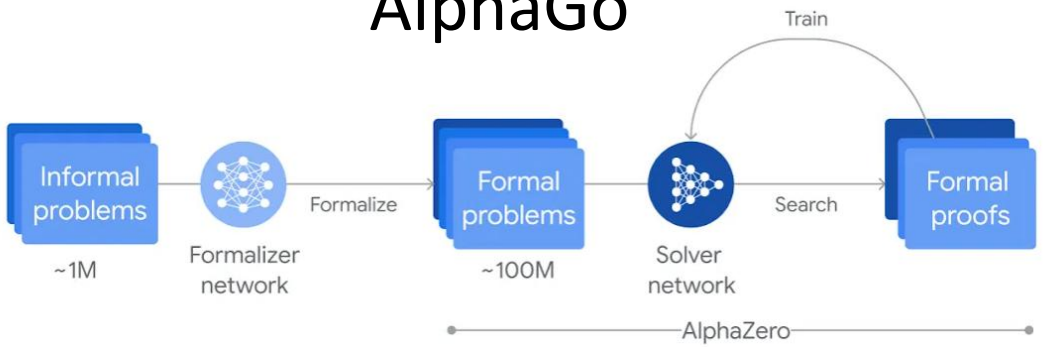
Why Reinforcement Learning

Some of the AI breakthroughs in the past **10 years**

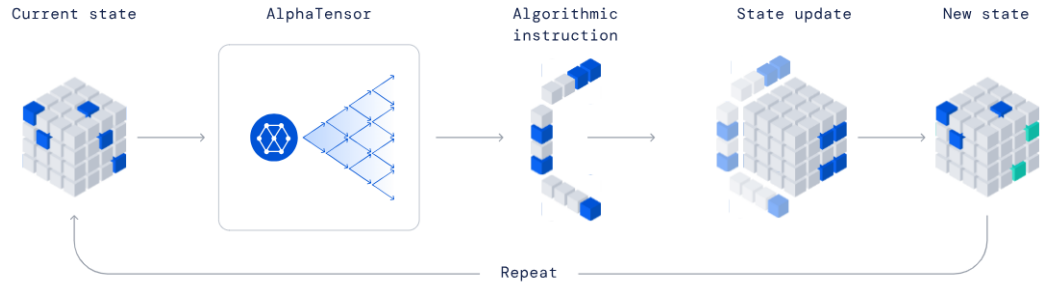


AlphaGo

AlphaStar



AlphaProof



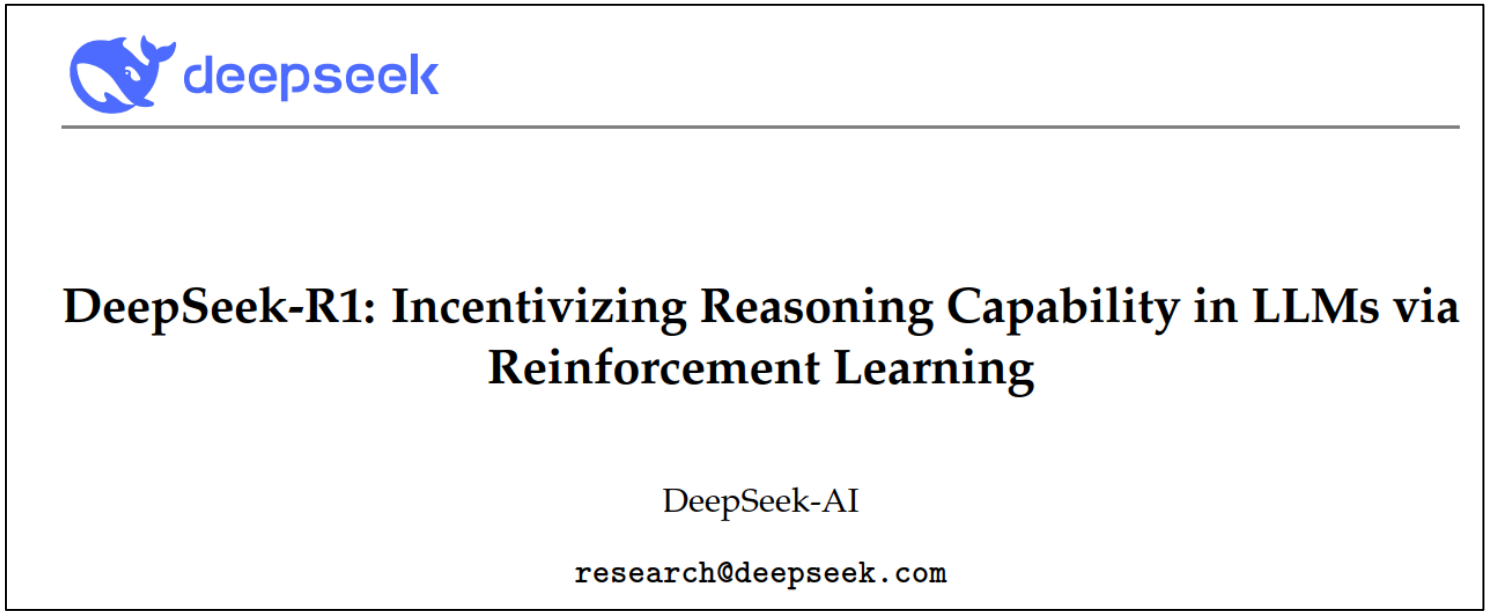
AlphaTensor

Why Reinforcement Learning

Some of the AI breakthroughs in the past **half year**

Our large-scale reinforcement learning algorithm teaches the model how to think productively using its chain of thought in a highly data-efficient training process. We have found that the performance of o1 consistently improves with more reinforcement learning (train-time compute) and with more time spent thinking (test-time compute). The

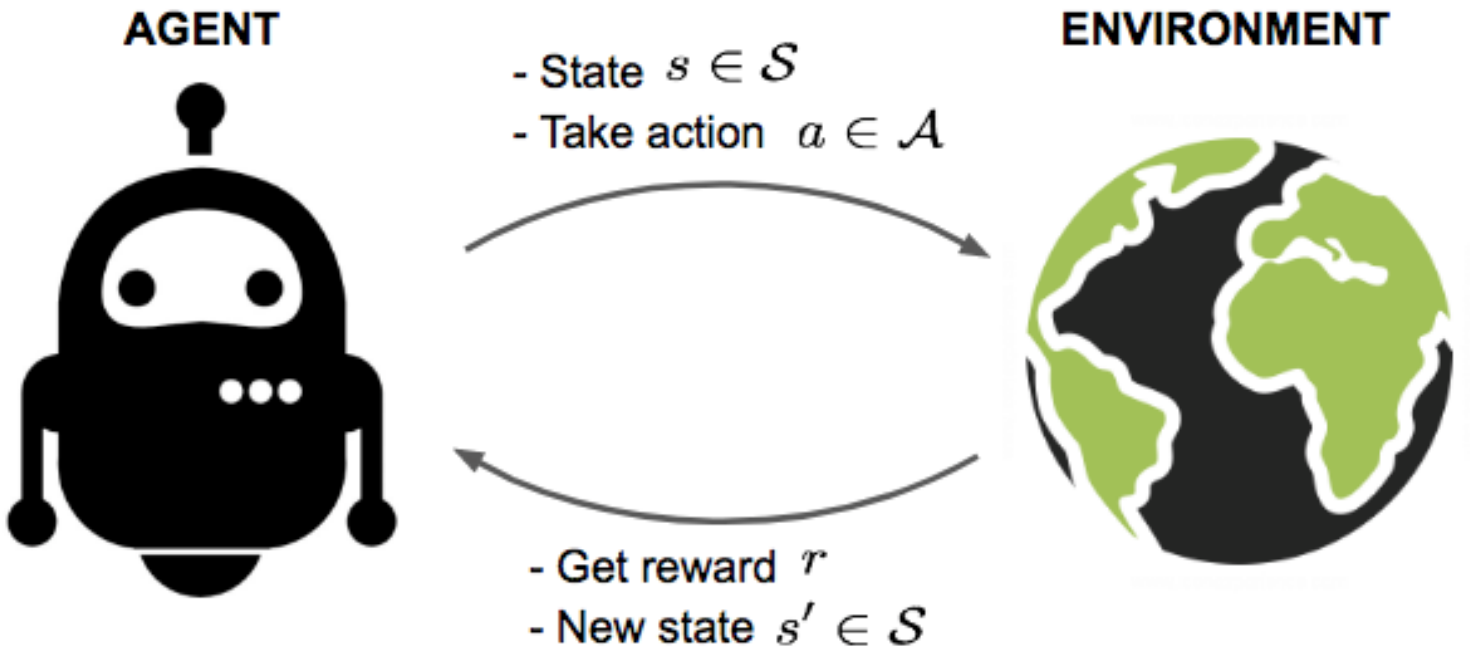
OpenAI o1



DeepSeek R1

Reinforcement Learning

The **agent** takes **actions** in an **environment** to **maximize** cumulative **rewards**



https://lilianweng.github.io/posts/2018-02-19-rl-overview/

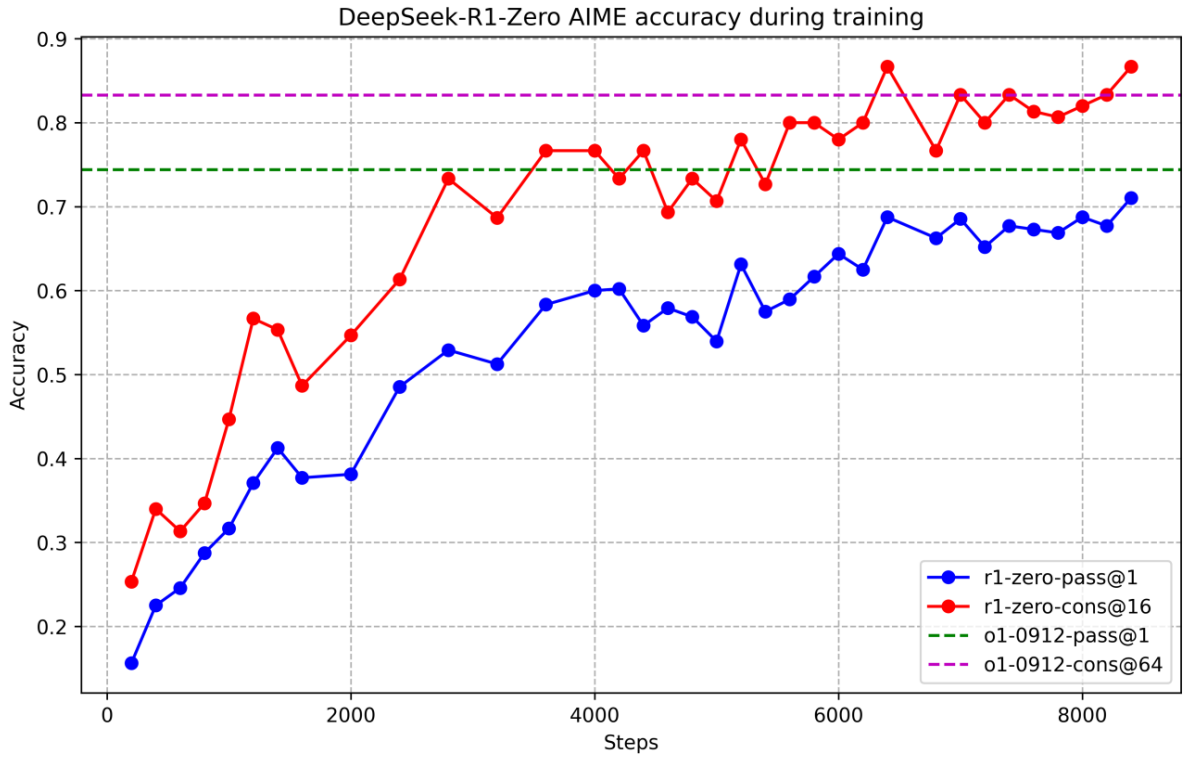
Reinforcement Learning

Key factors in scalable RL for LLMs

- A strong base policy
  - DeepSeek-V3 671B
- Unhackable, accurate rewards

$$r_o^{\text{math}}(\mathbf{y}) = \begin{cases} 1, & \text{matched} \\ 0, & \text{otherwise} \end{cases}$$

- Simple policy gradient works well
  - GRPO ≈ REINFORCE + Avg. as baseline

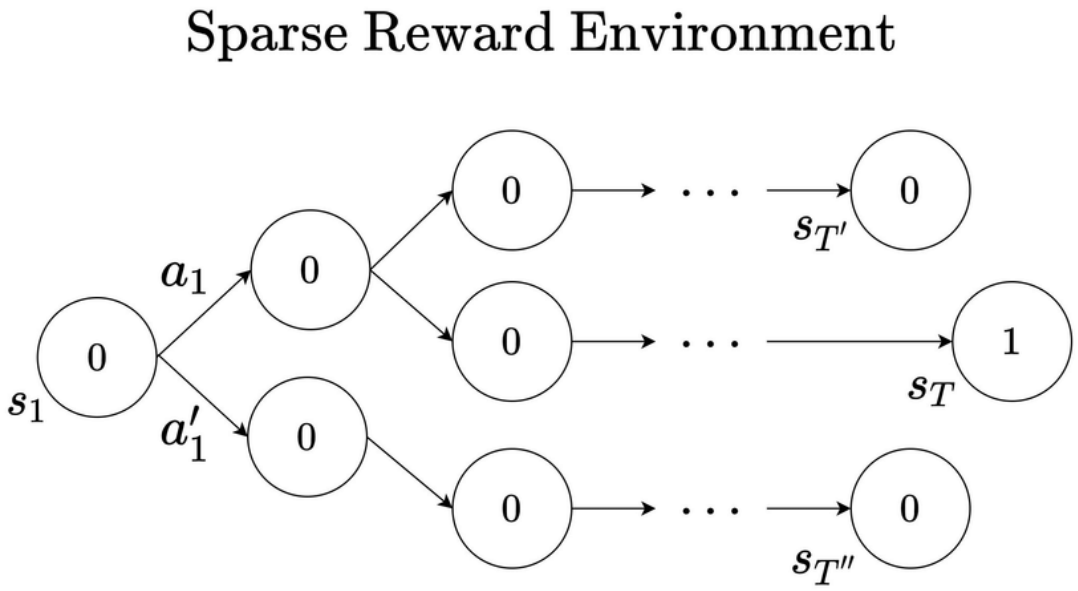


Guo et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. 2025

Reinforcement Learning

A missing part: **Dense Rewards**

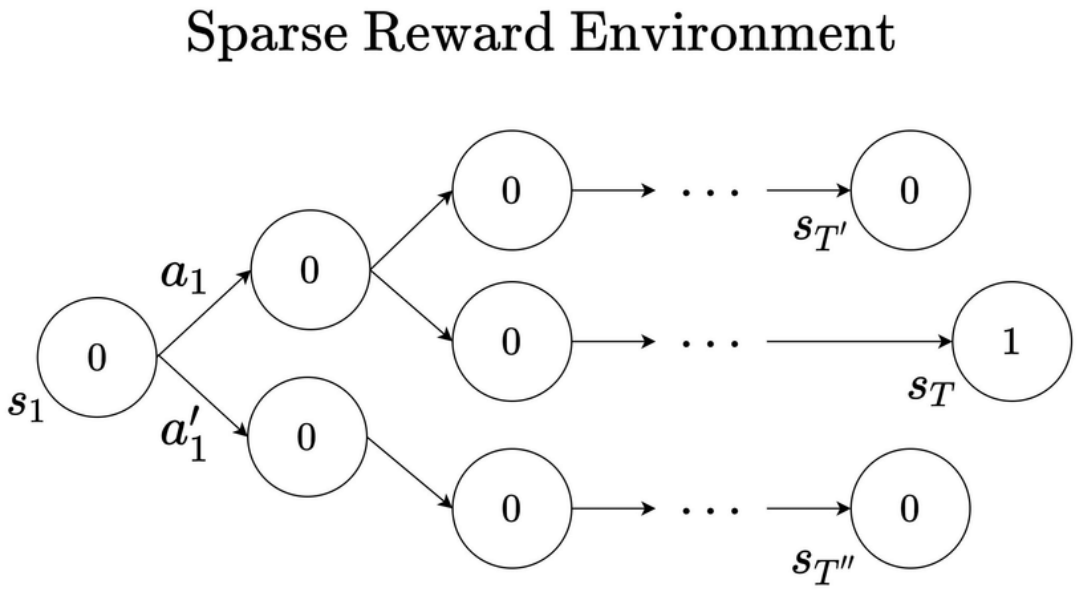
- Rule-based rewards / Outcome reward models only provide final rewards for responses
- Reward Sparsity



Reinforcement Learning

A missing part: **Dense Rewards**

- Rule-based rewards / Outcome reward models only provide final rewards for responses
- Reward Sparsity -> **Process Reward Model!**



Supervise Process, Not Outcome

– Andreas Stuhlmüller et al.

Let

$$x^8 + 3x^4 - 4 = p_1(x)p_2(x)\cdots p_k(x),$$

where each non-constant polynomial $p_i(x)$ is monic with integer coefficients, and cannot be factored further over the integers. Compute $p_1(1) + p_2(1) + \cdots + p_k(1)$.

This problem is about factoring a polynomial over the integers, which can be a tricky task.

One possible strategy is to look for rational roots of the polynomial, using the rational root theorem.

However, that would involve checking a lot of possible factors of -4, and there is no guarantee that the polynomial has any rational roots at all.

Another possible strategy is to use some algebraic tricks to simplify the polynomial or reduce its degree.

I notice that the polynomial has only even powers of x, so I can try to substitute $y = x^2$ and see what happens.

Then the polynomial becomes $y^4 + 3y^2 - 4$, which is a quadratic equation in y^2.

I can try to factor this quadratic equation by looking for two numbers that multiply to -4 and add to 3.

Those numbers are 4 and -1, so I can write $y^4 + 3y^2 - 4 = (y^2 + 4)(y^2 - 1)$.

Now I can substitute back $x^2 = y$ and get $x^8 + 3x^4 - 4 = (x^4 + 4)(x^4 - 1)$.

This looks promising, because I have reduced the degree of the factors from 8 to 4.

I can try to factor each of these factors further, using the difference of squares formula.

I get $x^8 + 3x^4 - 4 = (x^2 + 2)(x^2 - 2)(x^2 + 1)(x^2 - 1)$.

I can apply the difference of squares formula again to the last factor and get $x^8 + 3x^4 - 4 = (x^2 + 2)(x^2 - 2)(x^2 + 1)(x + 1)(x - 1)$.

Now I have factored the polynomial completely into monic linear and quadratic factors with integer coefficients.

These are the $p_i(x)$'s that the problem is asking for.

To find the sum of their values at $x = 1$, I just need to plug in $x = 1$ into each factor and add them up.

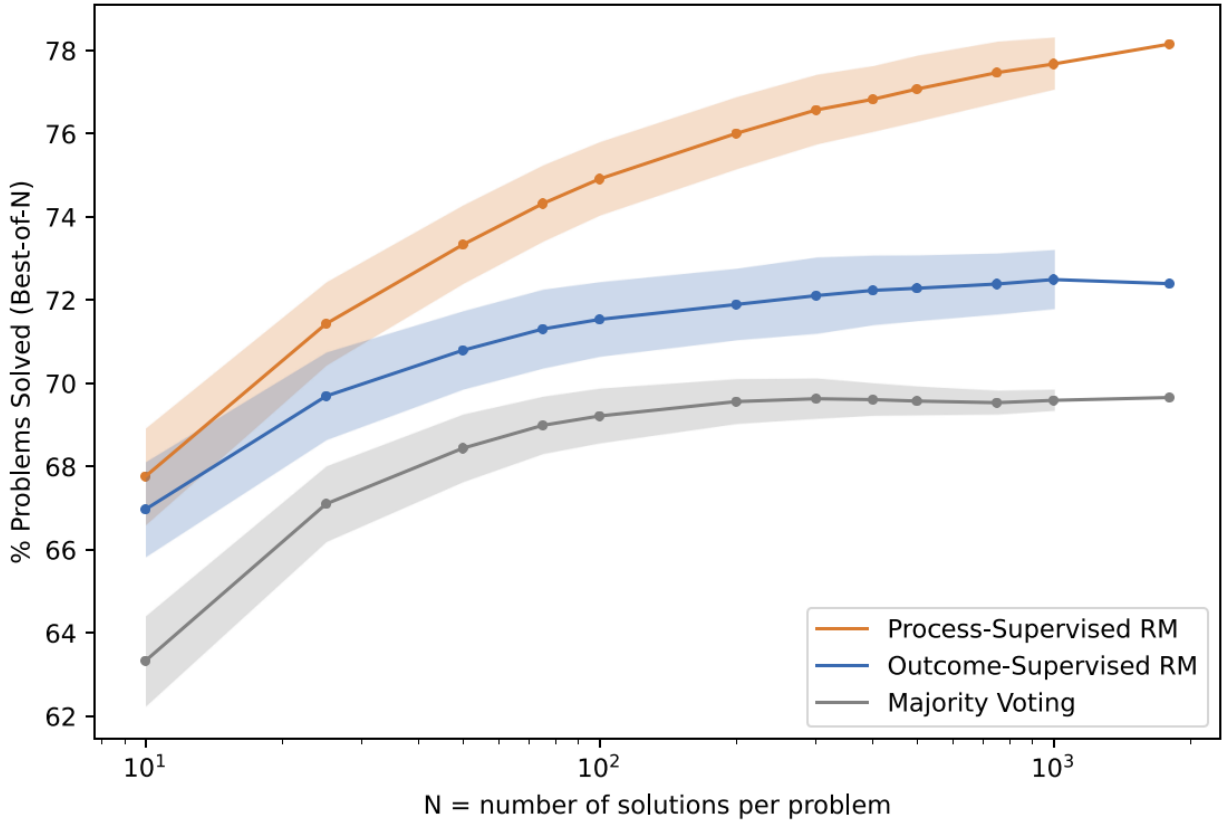
I get $p_1(1) + p_2(1) + \cdots + p_k(1) = (1^2 + 2)(1^2 - 2)(1^2 + 1)(1 + 1)(1 - 1)$.

Simplifying, I get $p_1(1) + p_2(1) + \cdots + p_k(1) = (3)(-1)(2)(2)(0)$.

Multiplying, I get $p_1(1) + p_2(1) + \cdots + p_k(1) = 0$.

Answer: 0

	ORM	PRM	Majority Voting
% Solved (Best-of-1860)	72.4	**78.2**	69.6



Lightman et al. Let's Verify Step by Step

How about PRM for RL?

No successful attempts!

3.3 Modeling Partial Completions is Not Necessary

As described in Sect. 2, PPO models each token as an action whereas REINFORCE models the *entire generation* as a single action, as opposed to each token. In practice, in LLM RLHF a $r(x, y)$ is only attributed to the <EOS> token, where for all other tokens, only $\log \frac{\pi(y_t|s_t)}{\pi_{\text{ref}}(y_t|s_t)}$ composes $R_t(x, y)$, which is not meaningful.

As the value function employed in PPO is typically another model of comparable size as the policy model, it brings a substantial memory and computational burden. Additionally, during RL training, the value function is treated as a baseline in the calculation of the advantage for variance reduction. While in the LLM context, usually only the last token is assigned a reward score by the reward model, which may complicate the training of a value function that is accurate at each token. To address this, as shown in Figure 4, we propose Group Relative Policy

We do not apply the outcome or process neural reward model in developing DeepSeek-R1-Zero, because we find that the neural reward model may suffer from reward hacking in the large-scale reinforcement learning process, and retraining the reward model needs additional training resources and it complicates the whole training pipeline.

https://arxiv.org/abs/2402.03300
https://arxiv.org/abs/2402.14740

How about RL?

Is RL with PRMs a dead end?

4.2. Unsuccessful Attempts

In the early stages of developing DeepSeek-R1, we also encountered failures and setbacks along the way. We share our failure experiences here to provide insights, but this does not imply that these approaches are incapable of developing effective reasoning models.

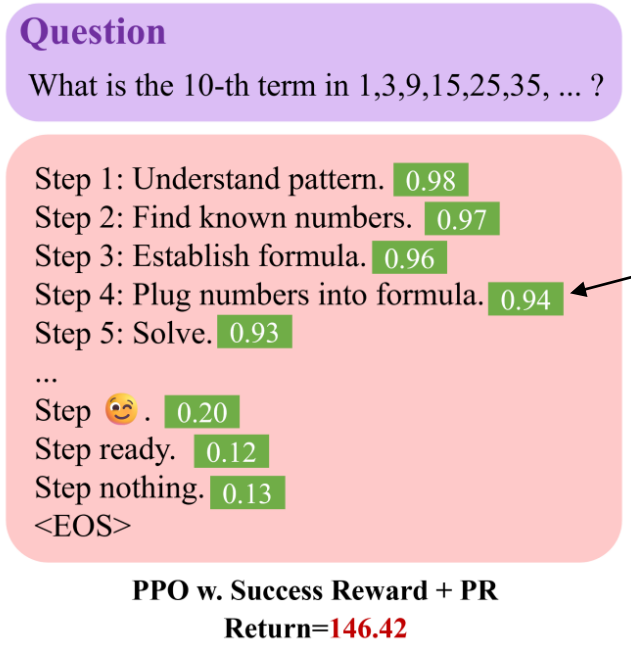
Process Reward Model (PRM) PRM is a reasonable method to guide the model toward better approaches for solving reasoning tasks (Lightman et al., 2023; Uesato et al., 2022; Wang et al., 2023). However, in practice, PRM has three main limitations that may hinder its ultimate success. First, it is challenging to explicitly define a fine-grain step in general reasoning. Second, determining whether the current intermediate step is correct is a challenging task. Automated annotation using models may not yield satisfactory results, while manual annotation is not conducive to scaling up. Third, once a model-based PRM is introduced, it inevitably leads to reward hacking (Gao et al., 2022), and retraining the reward model needs additional training resources and it complicates the whole training pipeline. In conclusion, while PRM demonstrates a good ability to rerank the top-N responses generated by the model or assist in guided search (Snell et al., 2024), its advantages are limited compared to the additional computational overhead it introduces during large-scale reinforcement learning process in our experiments.

Guo et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. 2025

Why is it hard to use PRM in RL?

Challenge 1: How to define steps and process rewards?

- Steps does not naturally occur in sequences
- Defining the **absolute correctness** of intermediate processes can be ambiguous



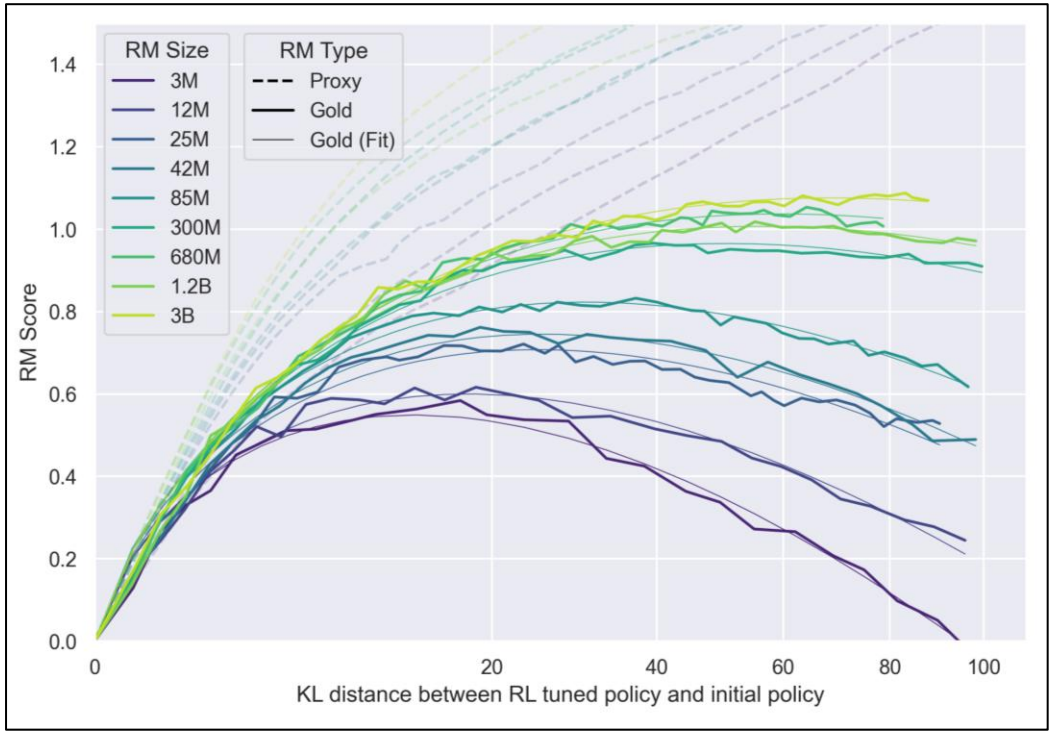
Policy model could simply repeat some "correct" nonsense to get high process rewards

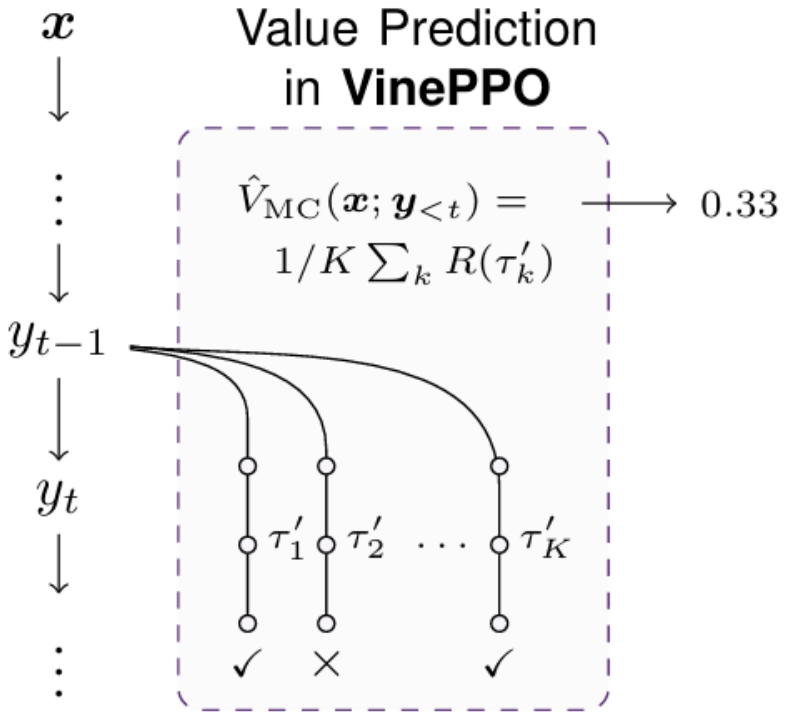
Gao et al. ON DESIGNING EFFECTIVE RL REWARD AT TRAINING TIME FOR LLM REASONING

Why is it hard to use PRM in RL?

Challenge 2: PRM online updates are not scalable

- Static RM/PRM inevitably leads to **reward hacking**
- Updating PRMs online is expensive, requires **50x** more rollouts





Gao et al. Scaling laws for reward model overoptimization. In ICML, 2022.
Kazemnejad et al. Vineppo: Unlocking rl potential for llm reasoning through refined credit assignment. 2024

Data Collection of PRM

Score intermediate steps one-by-one

- Expensive!

The denominator of a fraction is 7 less than 3 times the numerator. If the fraction is equivalent to $2/5$, what is the numerator of the fraction? (Answer: $\boxed{14}$)

☹ 😐 😎 Let's call the numerator x.

☹ 😐 😎 So the denominator is 3x-7.

☹ 😐 😎 We know that x/(3x-7) = 2/5.

☹ 😐 😎 So 5x = 2(3x-7).

☹ 😐 😎 5x = 6x - 14.

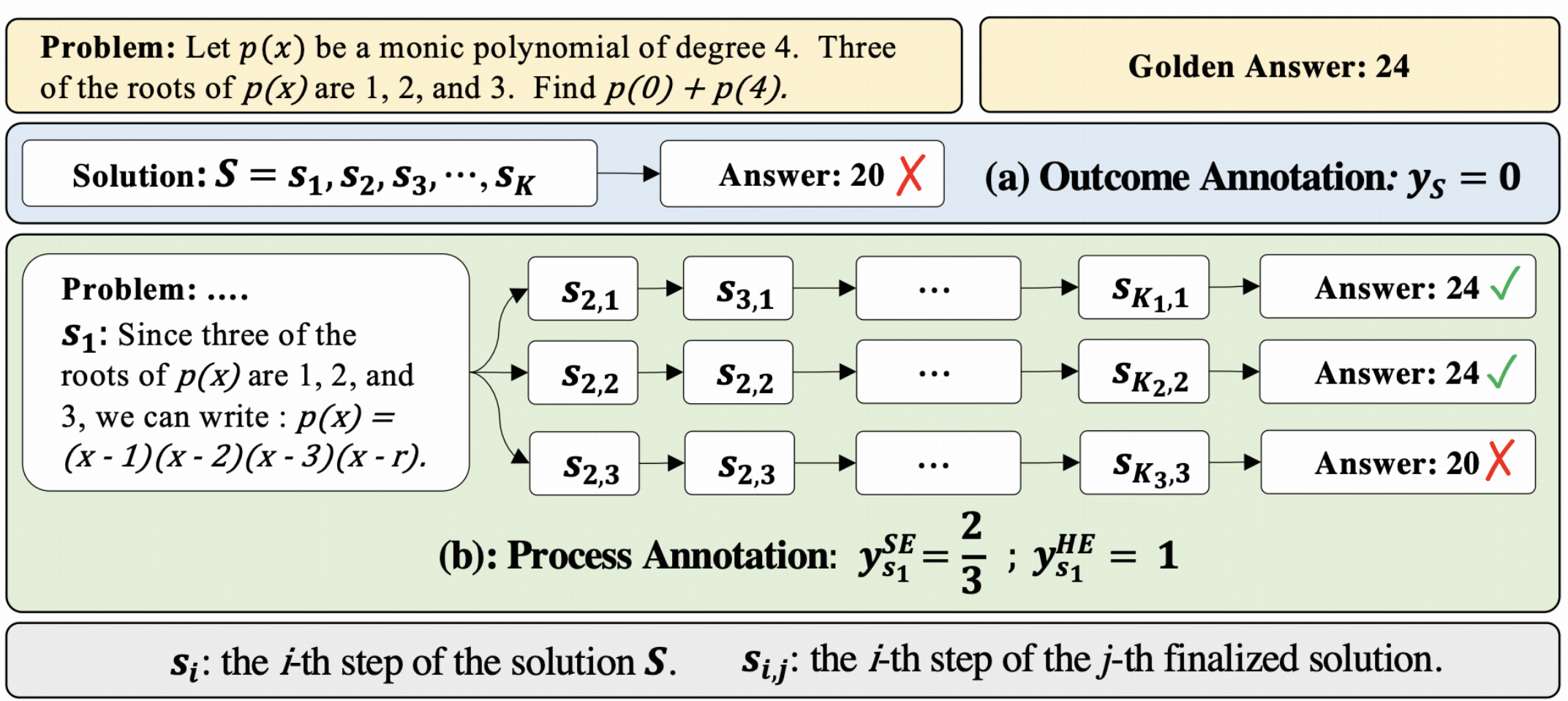
☹ 😐 😎 So x = 7.

Lightman et al. Let's Verify Step by Step. 2023

Data Collection of PRM

Automatic estimation with MCTS

- Also expensive!



Wang et al. MATH-SHEPHERD: A LABEL-FREE STEP-BY-STEP VERIFIER FOR LLMS IN MATHEMATICAL REASONING. 2023

Implicit PRM: Core Proposition

- We only need to do reward reparameterization, the learned ORM will automatically become a PRM

✨ ***Proposition:*** *Consider an ORM where the reward is parameterized by the log-likelihood ratio of two causal LMs, i.e.* $r_\phi(\mathbf{y}) := \beta \log \frac{\pi_\phi(\mathbf{y})}{\pi_{\text{ref}}(\mathbf{y})}$. *Define* $q_\phi^t(\mathbf{y}_{<t}, y_t) := \sum_{i=1}^{t} \beta \log \frac{\pi_\phi(y_i|\mathbf{y}_{<i})}{\pi_{\text{ref}}(y_i|\mathbf{y}_{<i})}$. q_ϕ^t *is the exponential average of* r_ϕ *at step* t.

$$q_\phi^t(\mathbf{y}_{<t}, y_t) = \beta \log \mathbb{E}_{\pi_{\text{ref}}(\mathbf{y}|\mathbf{y}_{\le t})} e^{\frac{1}{\beta} r_\phi(\mathbf{y})}$$

Hence, q_ϕ^t ***represents an exact expectation of outcome reward*** r_ϕ ***at step*** t***, i.e., the Q value.***

Define outcome reward as **log-likelihood ratio**

Get **closed-form solution** of Q-value

The proposition indicates that when modeling $r_\phi(\mathbf{y}) := \beta \log \frac{\pi_\phi(\mathbf{y})}{\pi_{\text{ref}}(\mathbf{y})}$ to train an ORM with the standard pipeline, where β is a hyperparameter, ϕ can implicitly learn a Q function. Hence, process reward r_ϕ^t can be obtained by:

$$r_\phi^t := q_\phi^t - q_\phi^{t-1} = \beta \log \frac{\pi_\phi(y_t|\mathbf{y}_{<t})}{\pi_{\text{ref}}(y_t|\mathbf{y}_{<t})}$$

Get **free process rewards** as Q-value diff

Yuan et al. FREE PROCESS REWARDS WITHOUT PROCESS LABELS. 2024

Train as an ORM

x

$\mathbf{y}$

l

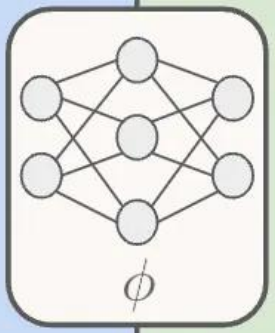
Forward pass on **ORM data** to obtain implicit reward

$$r_\phi(\mathbf{y}) := \beta \log \frac{\pi_\phi(\mathbf{y})}{\pi_{\text{ref}}(\mathbf{y})}$$

$$\mathcal{L}_{\text{CE}}\left(r_\phi\left(\mathbf{y}\right), l\right)$$

Train an ORM with vanilla ORM loss (e.g., CE loss)

Outcome label



Inference as an Implicit PRM

x'

y'_1 ... y'_t ... y'_n

Log-likelihood ratio on partial response represents Q value:

$$q_\phi^t(\mathbf{y}_{<t}, y_t) := \sum_{i=1}^{t} \beta \log \frac{\pi_\phi(y_i|\mathbf{y}_{<i})}{\pi_{\text{ref}}(y_i|\mathbf{y}_{<i})}$$

Derive implicit process reward from Q value:

$$r_\phi^t(\mathbf{y}) := \beta \log \frac{\pi_\phi(y_t|\mathbf{y}_{<t})}{\pi_{\text{ref}}(y_t|\mathbf{y}_{<t}))}$$

$$\mathbf{y}' = y'_1 y'_2 \cdots y'_t \cdots y'_n$$

y'_t can be either a single token or a step

Implicit PRM

Implicit PRM: Obtain process rewards through **outcome reward modeling** at **no additional cost**!

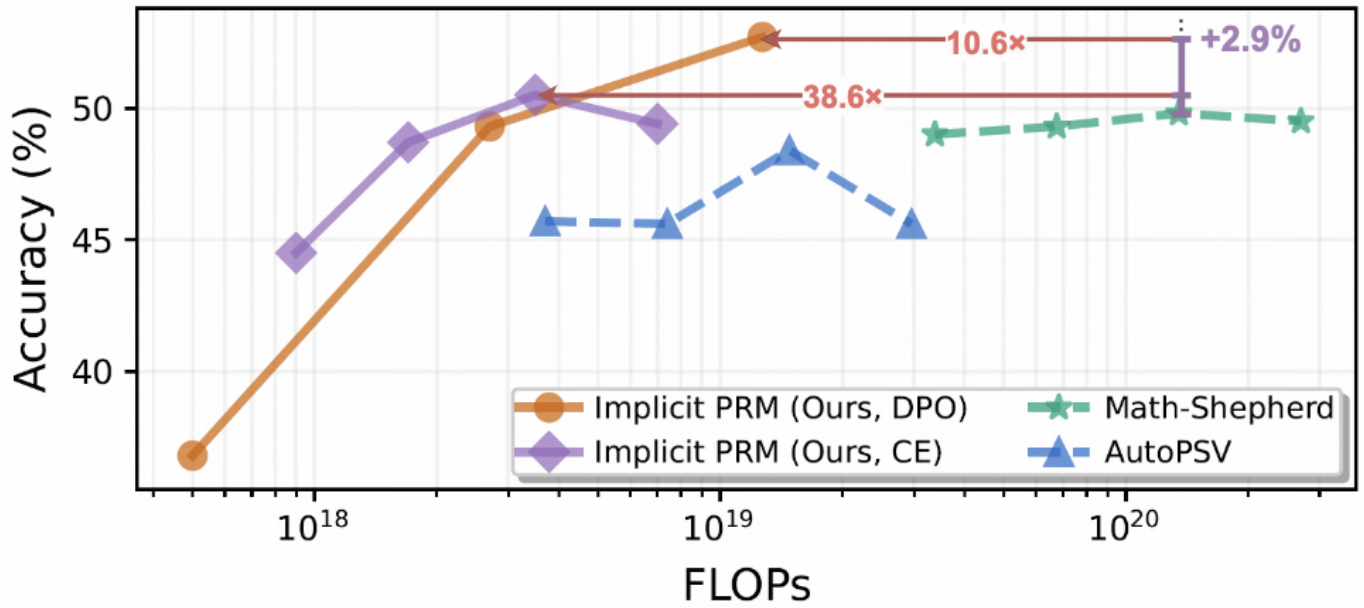


Figure 1: The x-axis indicates the FLOPs required to collect the data and train the model, and y axis the accuracies of best-of-64 performance. The accuracy is averaged over the best-of-64 accuracies of Mistral-7B-Instruct-v0.2 (Jiang et al., 2023), Llama-3.1-8B-Instruct, and Llama-3.1-70B-Instruct (Meta, 2024) on MATH (Hendrycks et al., 2021). Different dots on the same line indicates models trained with the same approach but on different scales of data. The top-left zone is desirable in this figure, as it suggests a model can achieve higher performance with less development overhead. Our implicit PRM is much cheaper to train while presenting the best performance under the same budget.

Yuan et al. FREE PROCESS REWARDS WITHOUT PROCESS LABELS. 2024

Implicit PRM

Solution 1: Rewarding progress, Not correctness

- Implicit PRM enables tractable Q value calculation for **each token** at no cost
- Consider process reward as **relative** advantage

$$r_\phi^t := q_\phi^t - q_\phi^{t-1} = \beta \log \frac{\pi_\phi(y_t|\mathbf{y}_{<t})}{\pi_{\text{ref}}(y_t|\mathbf{y}_{<t})}$$

Solution 2: Train on outcome labels

- Implicit PRM only needs outcome labels to update
- Converts **any** sparse outcome reward into dense process rewards

The PRIME

Integrating Implicit PRM into RL

- Basic policy gradient (REINFORCE)

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(y_t|\mathbf{y}_{<t})A_t\right]$$

- Advantage estimation

$$A_t = \sum_{s=t}^{T}\gamma^{s-t}r(y_s) - b$$

The PRIME

Integrating Implicit PRM into RL

- REINFORCE Leave-one-out (RLOO)

$$A^i = r(\mathbf{y}_T^i) - \frac{1}{K-1}\sum_{j\neq i} r(\mathbf{y}_T^j)$$

- RLOO with process rewards

$$A_t^i = \underbrace{\sum_{s=t}^{|\mathbf{y}^i|} \gamma^{s-t} \cdot \left[r_\phi(y_s^i) - \frac{1}{K-1}\sum_{j\neq i} r_\phi\left(\mathbf{y}^j\right) \right]}_{\text{RLOO with implicit process rewards}} + \underbrace{r_o\left(\mathbf{y}^i\right) - \frac{1}{K-1}\sum_{j\neq i} r_o\left(\mathbf{y}^j\right)}_{\text{RLOO with outcome rewards}}$$

The PRIME

Algorithm 1 Process Reinforcement through Implicit Rewards (PRIME)

Input Language model $\pi_{\theta_{\text{init}}}$; outcome reward verifier r_o; dataset $\mathcal{D}$; sample number K; total iteration N.

Initialize policy model $\pi_\theta \leftarrow \pi_{\theta_{\text{init}}}$, $\pi_{\theta_{\text{old}}} \leftarrow \pi_{\theta_{\text{init}}}$, implicit PRM $\pi_\phi \leftarrow \pi_{\theta_{\text{init}}}$, reference model $\pi_{\text{ref}} \leftarrow \pi_{\theta_{\text{init}}}$

for iteration = 1, . . . , N **do**

  Sample batch of prompts $\mathcal{B} \sim \mathcal{D}$

  Generate K responses: $\{\mathbf{y}^1, ..., \mathbf{y}^K\} \sim \pi_\theta(\cdot|\mathbf{x})$ for $\mathbf{x} \in \mathcal{B}$

  Compute outcome rewards: $r_o\left(\mathbf{y}^{1:K}\right)$

  Apply accuracy filter (§3.3) on all prompts: $\mathcal{T} \leftarrow \text{Filter}(\mathbf{x}, \mathbf{y}^{1:K}, r_o\left(\mathbf{y}^{1:K}\right))$ for $\mathbf{x} \in \mathcal{B}$

  Forward pass $\pi_\phi, \pi_{\text{ref}}$ on each $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}$ to obatin implicit process reward $r_\phi(y_t)$ with Eq. 3

  Update Implicit PRM π_ϕ by CE loss on $(\mathbf{x}, \mathbf{y}, r_o\left(\mathbf{y}\right)) \in \mathcal{T}$:

$$\mathcal{L}_{\text{CE}}(\phi) = -\mathbb{E}_{(\mathbf{x},\mathbf{y},r_o(\mathbf{y}))\sim\mathcal{T}}\left[r_o\left(\mathbf{y}\right) \cdot \log \sigma\left(r_\phi\left(\mathbf{y}\right)\right) + \left(1 - r_o\left(\mathbf{y}\right)\right) \cdot \log\left(1 - \sigma\left(r_\phi\left(\mathbf{y}\right)\right)\right)\right]$$

  Compute advantages A with Eq. 5

  Update policy π_θ by PPO loss in Eq. 6

  Update old parameters: $\theta_{\text{old}} \leftarrow \theta$

end for

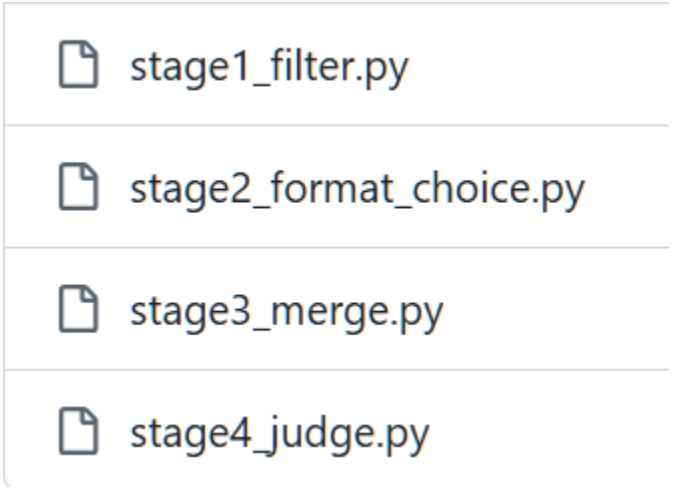
Output Optimized policy model π_θ

The PRIME

A simple, accessible, and powerful algo

Experiments

- A simple SFT warmup to initialize (may not be necessary)
- Collect top-quality open-source data **with verifiable rewards**
  - Math: NuminaMath-CoT
  - Code: APPS, CodeContests, TACO, Codeforces
- Four-stage data cleansing
  - Filter out questions for proof/figures/tables
  - Classify QA/MC/fill-in-the-blank
  - Reformulate MC questions into QA questions
  - Validate solution with reasoning models
    - Maj@5 with QwQ-32B



Experiments

Dense rewards vs Sparse rewards

- Process rewards are **2.5x more sample efficient** than outcome rewards!
- Also consistently outperform on testset

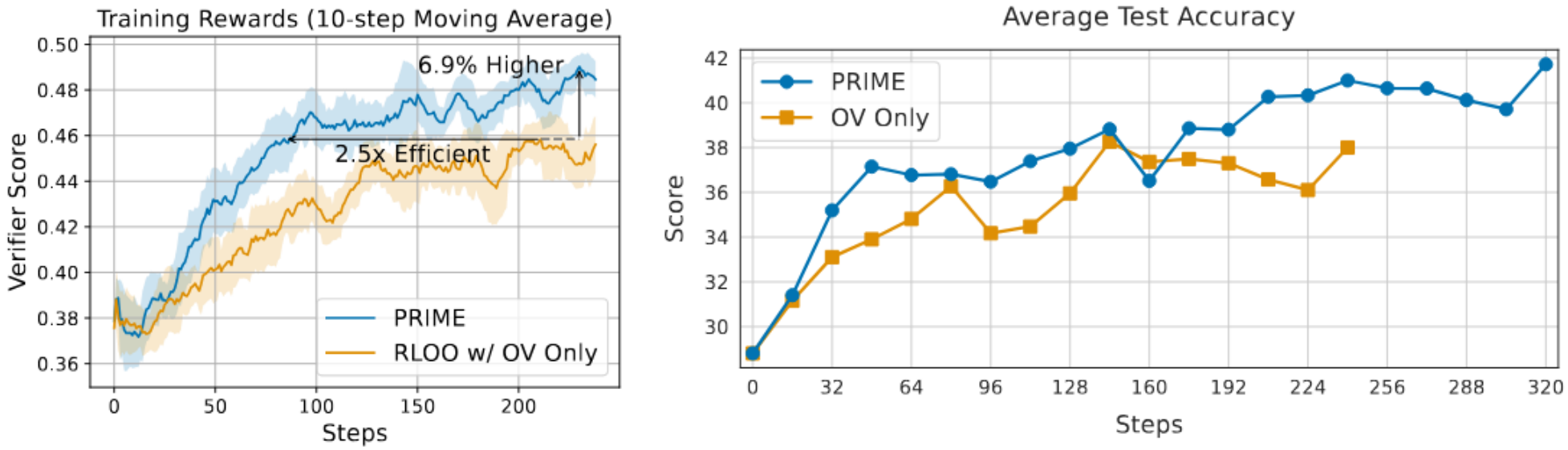


Figure 3: **The effect of dense reward.** We compare PRIME and RLOO with outcome verifier (OV). Dense rewards in PRIME lead to 2.5× sample efficiency and 6.9% performance improvement. PRIME also substantially outperforms RLOO on downstream tasks.

Experiments

Dense rewards vs Sparse rewards

- Process rewards are **2.5x more sample efficient** than outcome rewards!
- Also consistently outperform on testset

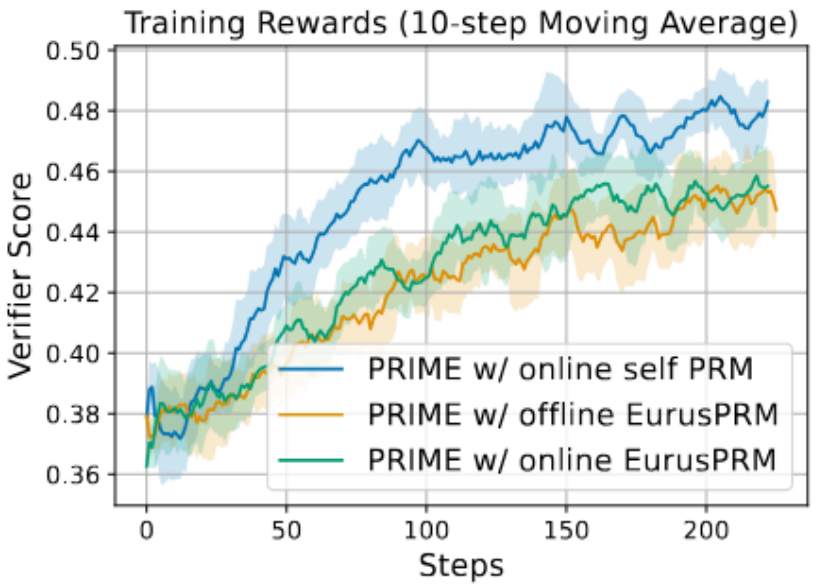
Table 2: Detailed results of PRIME and RLOO w/ outcome verifier (OV). At the same 240 steps, the model trained by PRIME is generally better than the model trained by outcome rewards.

Method	Step	AIME 2024	AMC	MATH-500	MinervaMath	OlympiadBench	LeetCode	LiveCodeBench	Avg.
GPT-4o	-	9.3	45.8	76.4	36.8	43.3	58.9	48.8	45.6
Llama-3.1-70B-Inst.	-	20.0	37.3	65.0	37.1	30.5	35.0	34.4	37.0
Qwen2.5-Math-7B-Inst.	-	13.3	50.6	79.8	34.6	40.7	11.7	11.3	34.6
Eurus-2-7B-SFT	0	3.3	30.1	66.2	32.7	29.8	21.7	17.8	28.8
RLOO w/ OV Only	240	**20.0**	47.0	73.2	36.4	35.4	28.3	26.7	36.9
Eurus-2-7B-PRIME	80	20.0	41.0	68.2	38.2	37.0	26.7	26.6	36.8
	160	13.3	42.2	72.0	37.1	38.7	26.7	25.6	36.5
	240	**20.0**	**50.6**	**78.2**	**39.3**	**40.3**	**31.1**	**27.5**	**41.0**
	320	16.7	51.8	77.8	39.7	41.5	36.1	28.5	41.7
	592	26.7	57.8	79.2	38.6	42.1	33.3	28.6	43.9

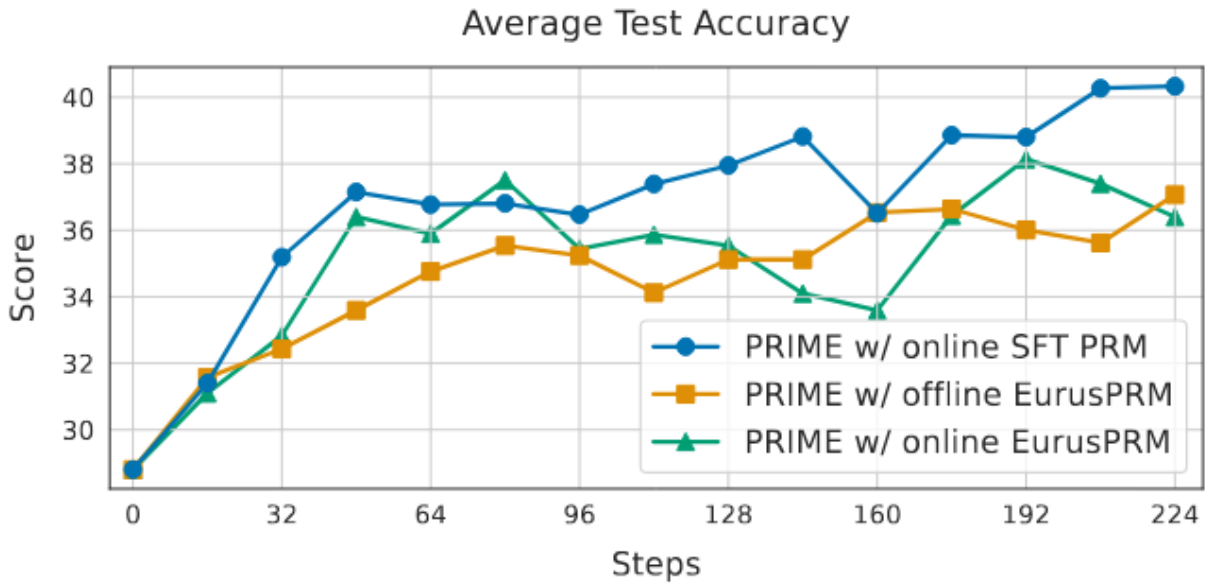
Experiments

Online PRM update

- Online PRM is substantially better than offline PRM



(a) Training outcome rewards.



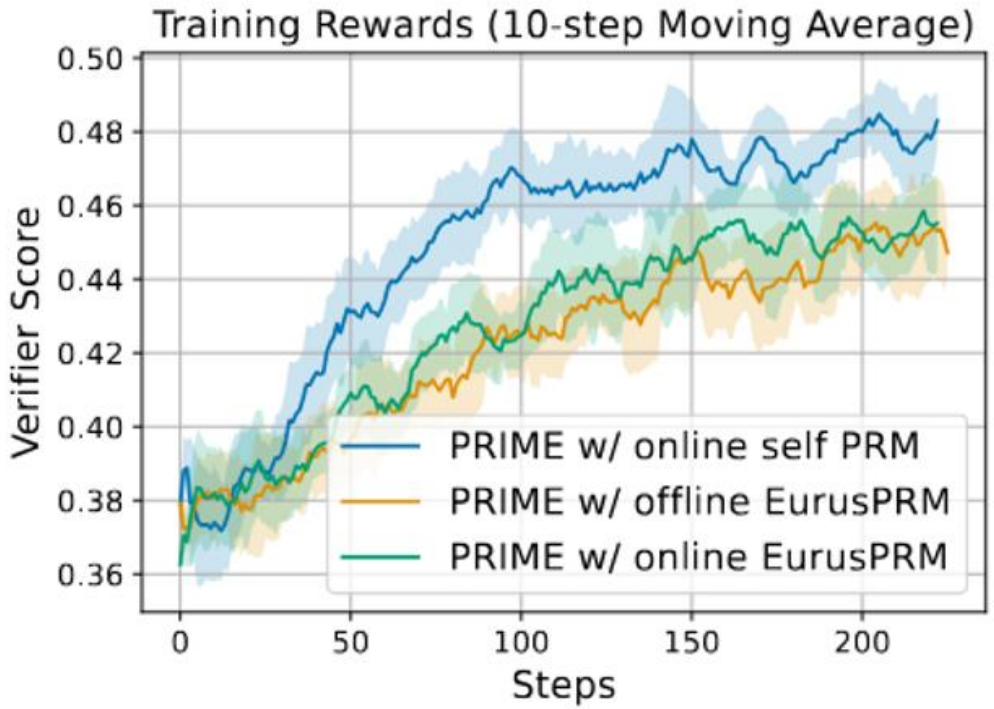
(b) Testset accuracy.

Figure 4: **Comparison of different PRMs.** Online PRM initialized from SFT model achieved the best results. Notably, using PRMs trained on extra rollouts caused performance degradation, in both online and offline settings.

Experiments

Online PRM update

- Online update is the game changer. Offline PRM eventually got overoptimized
- SFT model initializes a good PRM



(a) Training outcome rewards.

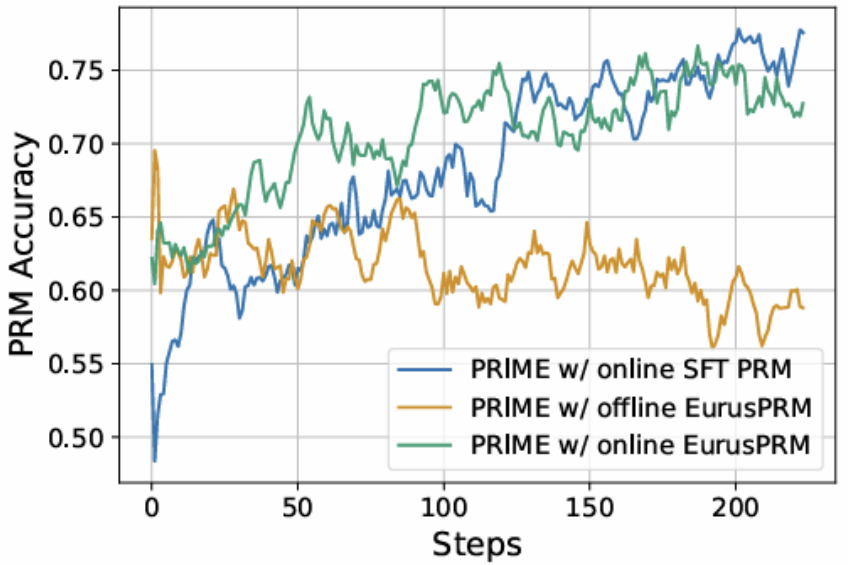


Figure 6: **Impact of PRM online update.** The offline PRM is gradully been overoptimized while online PRMs achieve higher accuracy throughout training.

Experiments

PRIME can work well with other RL algorithms

- GRPO

$$A_t^i = \underbrace{\frac{r_o\left(\mathbf{y}^i\right) - \text{mean}(r_o\left(\mathbf{y}^j\right))}{\text{std}(r_o\left(\mathbf{y}^j\right))}}_{\text{GRPO with outcome rewards}} + \underbrace{\sum_{s=t}^{|\mathbf{y}^i|} \gamma^{s-t} \cdot \left[\frac{r_\phi(y_s^i) - \text{mean}\left(\frac{r_\phi(\mathbf{y}^j)}{|\mathbf{y}^j|}\right)}{\text{std}\left(\frac{r_\phi(\mathbf{y}^j)}{|\mathbf{y}^j|}\right)}\right]}_{\text{GRPO with implicit process rewards}}.$$

- REINFORCE

$$A_t^i = \underbrace{r_o\left(\mathbf{y}^i\right)}_{\text{REINFORCE with outcome rewards}} + \underbrace{\sum_{s=t}^{|\mathbf{y}^i|} \gamma^{s-t} \cdot r_\phi(y_s^i)}_{\text{REINFORCE with implicit process rewards}}.$$

- PPO has an extra value model

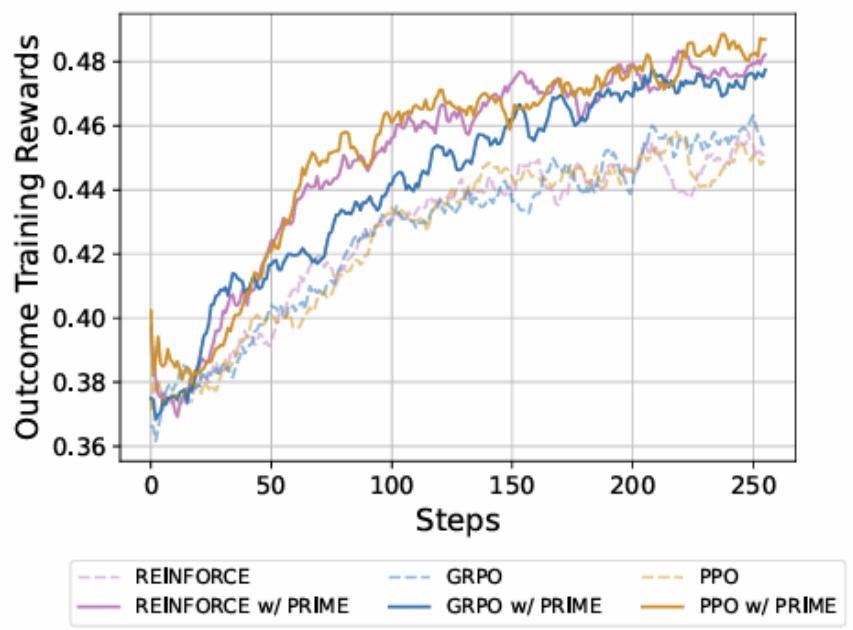


Figure 10: PRIME also benefits REINFORCE, GRPO, and PPO, achieving similar improvement as RLOO.

Experiments

Two different usages of Implicit PRM: Value or Reward?

- Key difference in **Return!**
- **Process reward is better than value**

(1) REINFORCE: $A_t = r_o(\mathbf{y})$.

(2) On top of (1), using **a linear-head value model** V to calculate the baseline: $A_t = r_o(\mathbf{y}) - V(\mathbf{y}_{<t})$. This is the original PPO in Figure 6 as we set $\gamma = 1$ and $\lambda = 1$.

(3) On top of (1), using **values from the Implicit PRM** to serve as the baseline: $A_t = r_o(\mathbf{y}) - v_\phi(\mathbf{y}_{<t})$. This is equivalent to PPO with its value model being replaced by values from the Implicit PRM when $\gamma = 1$ and $\lambda = 1$.

(4) On top of (1), using **process rewards from the Implicit PRM** to calculate the return: $A_t = r_o(\mathbf{y}) + \sum_{s=t}^{T} r_\phi(y_s)$. This is the REINFORCE w/ PRIME in Figure 6.

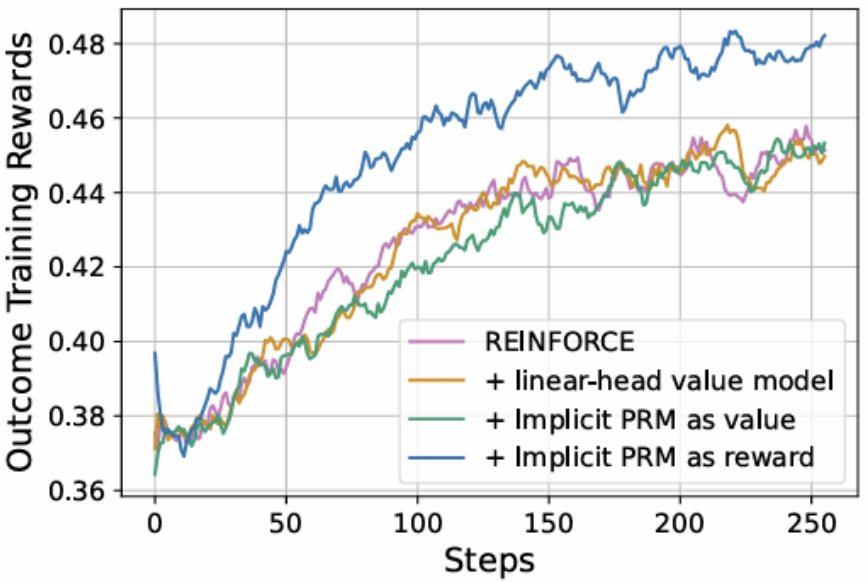


Figure 11: Comparison of value models and reward models. We show that value models, either the original PPO one or Implicit PRM, is substaintially worse than reward models.

Experiments

Overall comparison

- RLOO is the best performing
- PRIME is a general plug-in to almost all RL methods

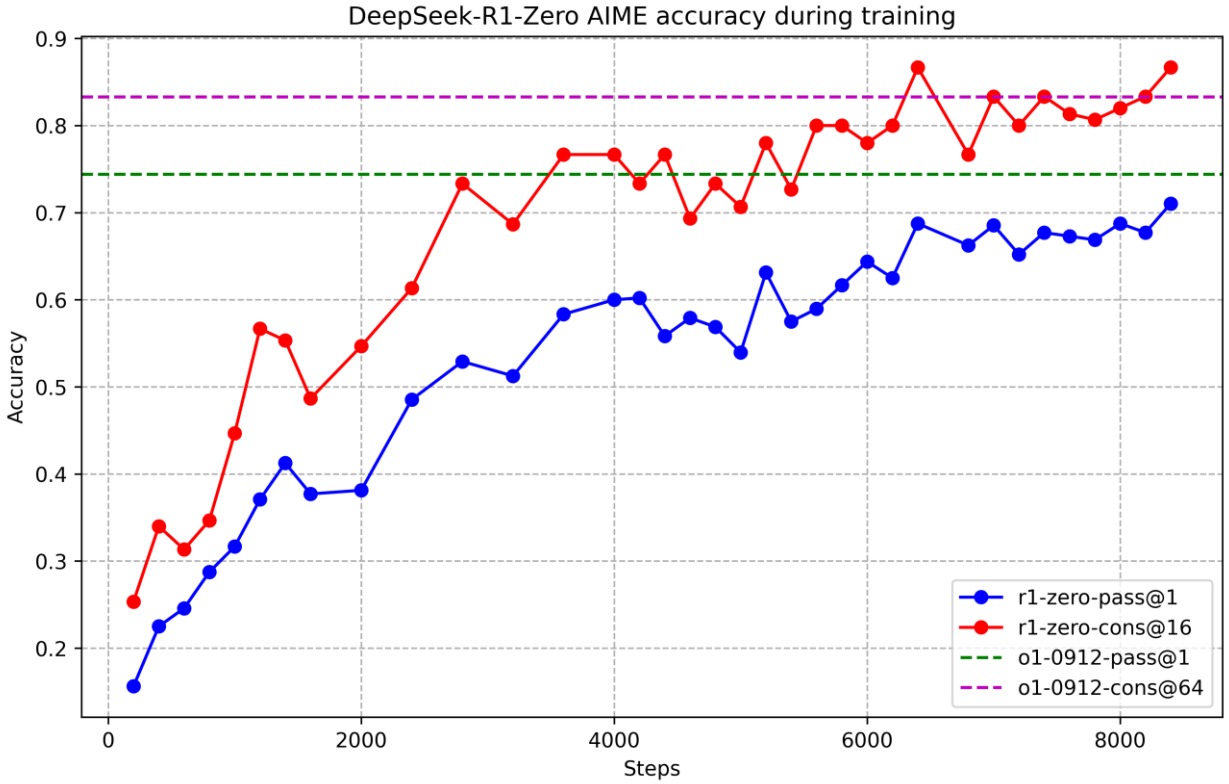
Table 3: Testset results of different RL algorithms.

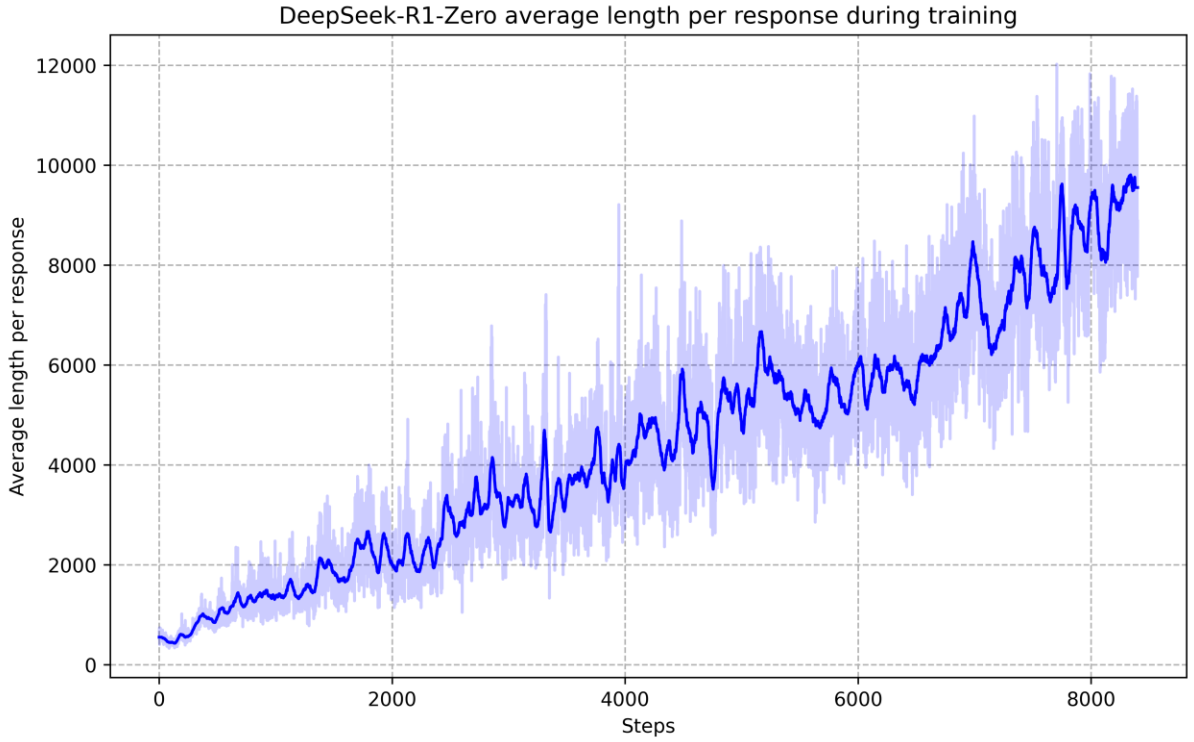
Method	Step	AIME 2024	AMC	MATH-500	MinervaMath	OlympiadBench	LeetCode	LiveCodeBench	Avg.
RLOO	240	20.0	47.0	73.2	36.4	35.4	28.3	26.7	36.9
RLOO w/ PRIME	240	20.0	50.6	78.2	39.3	40.3	31.1	27.5	41.0
REINFORCE	240	6.7	47.0	72.6	36.0	37.2	27.2	25.0	36.0
REINFORCE w/ PRIME	240	6.7	50.0	76.4	36.8	39.1	27.8	27.5	37.8
GRPO	240	10.0	44.6	73.2	37.5	36.6	25.0	25.8	36.1
GRPO w/ PRIME	240	16.7	47.0	75.0	34.9	38.2	28.9	23.9	37.8
PPO	240	10.0	41.0	73.6	36.0	36.3	28.3	25.7	35.8
PRIME as Value Model	240	16.7	44.6	72.6	34.6	35.7	27.8	24.6	36.6
PPO w/ PRIME	240	13.3	50.6	77.4	37.1	40.6	30.0	26.7	39.4

“Zero” Experiments

DeepSeek-R1-Zero

- RL directly from the base model
- Surprisingly effective

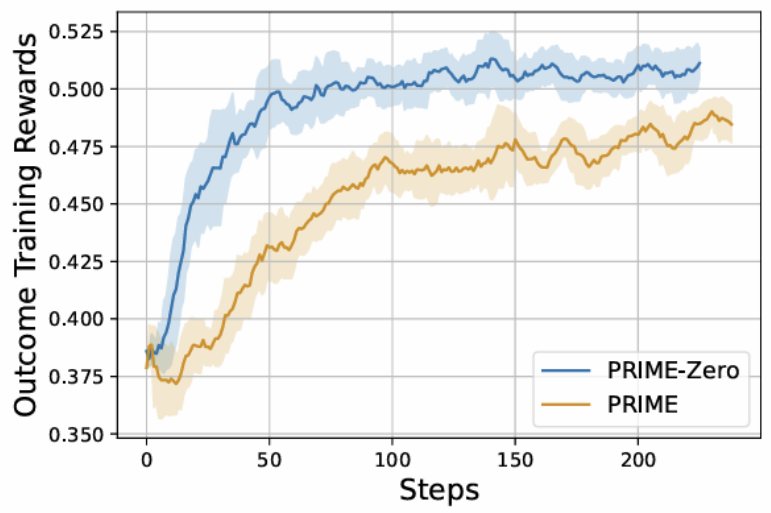




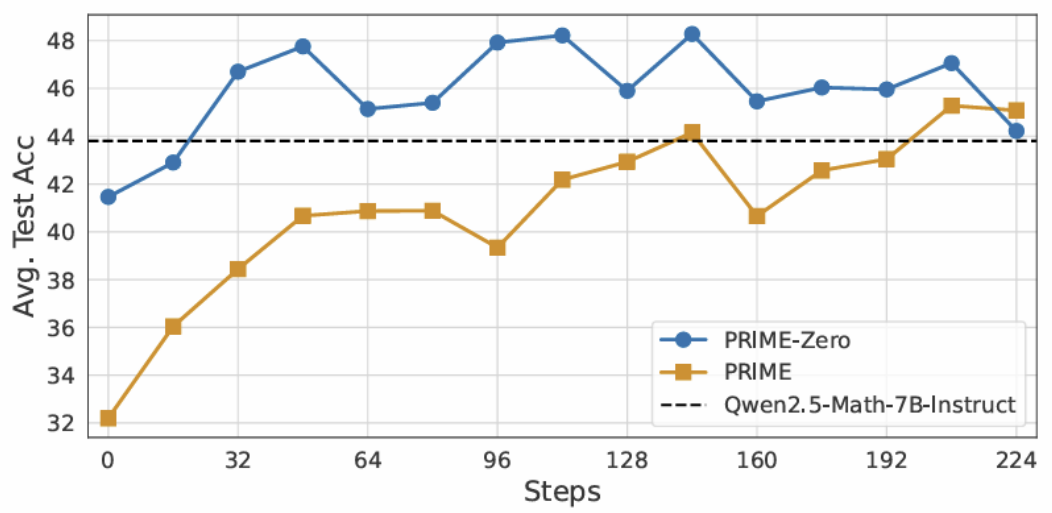
“Zero” Experiments

Zero RL from Qwen-2.5-Math-7B-Base

- Really **efficient**, but quickly **saturate**



(a) Outcome training rewards (10-step moving).



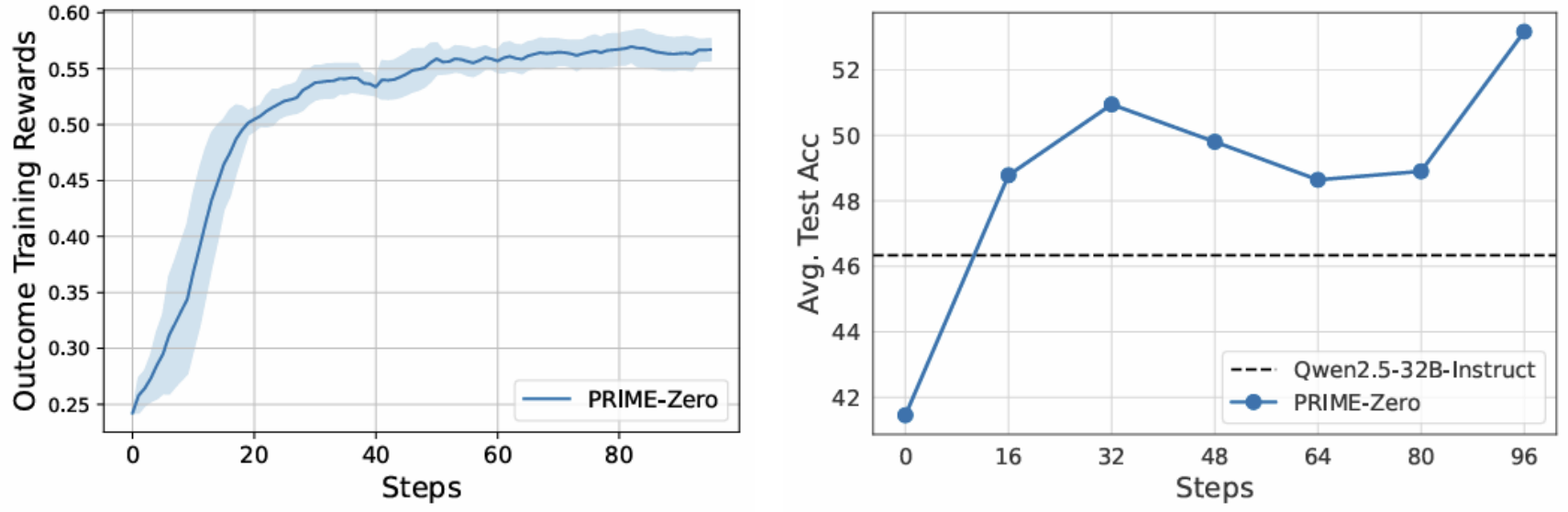
(b) Math test accuracy across different gradient steps.

Figure 12: **“Zero” RL from Qwen2.5-Math-7B.** RL from the base model converges way faster than the SFT model, surpassing the instruct version within 32 steps.

“Zero” Experiments

Zero RL from Qwen-2.5-32B-Base

- Larger models benefit more, and saturate slower

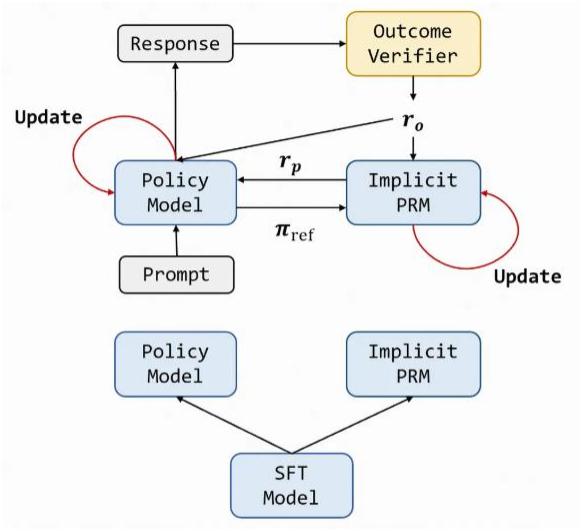


(a) Outcome training rewards (10-step moving). (b) Math test accuracy across different gradient steps.

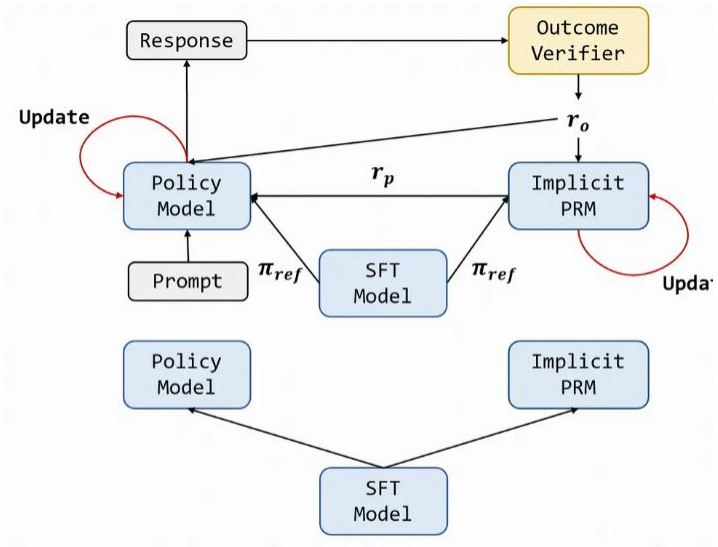
Figure 13: **“Zero” RL from Qwen2.5-32B-Base.** RL from a 32B base model shows more promising gain, surpassing the instruct version within 16 steps.

Experiments

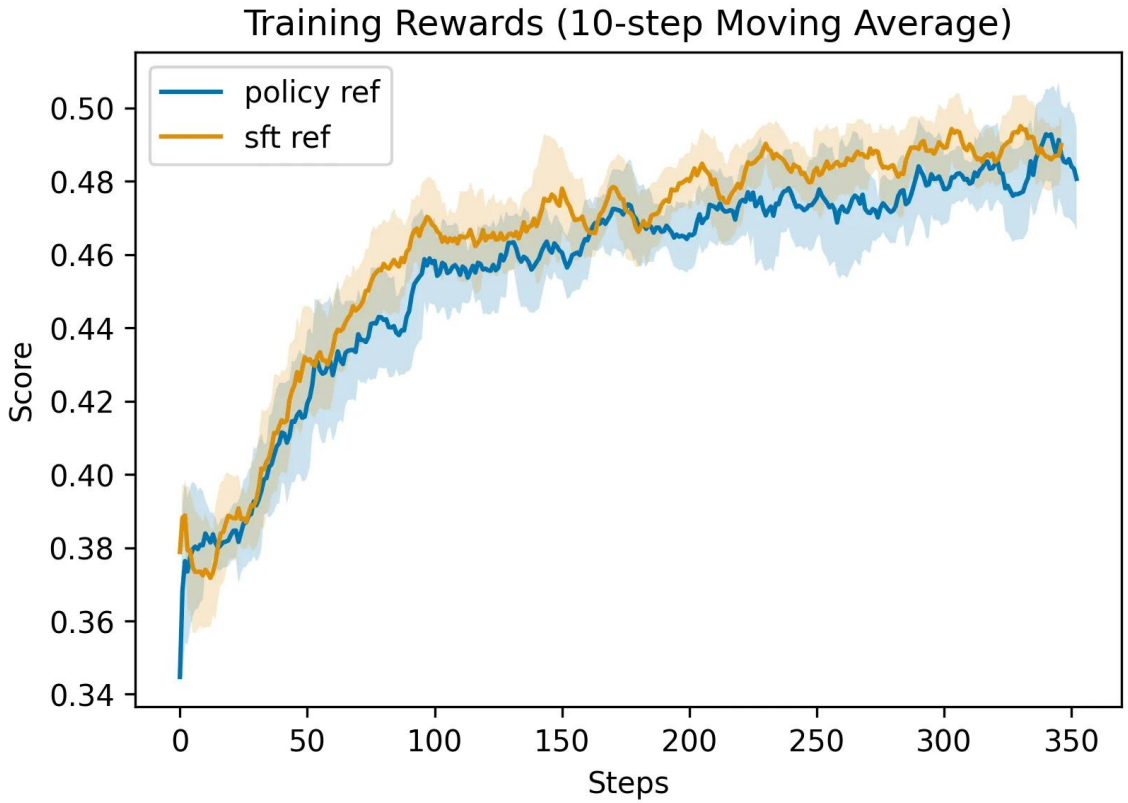
Reference model choice is flexible



Policy ref: We discard the reference policy and use the old logprob as π_{ref} for PRM

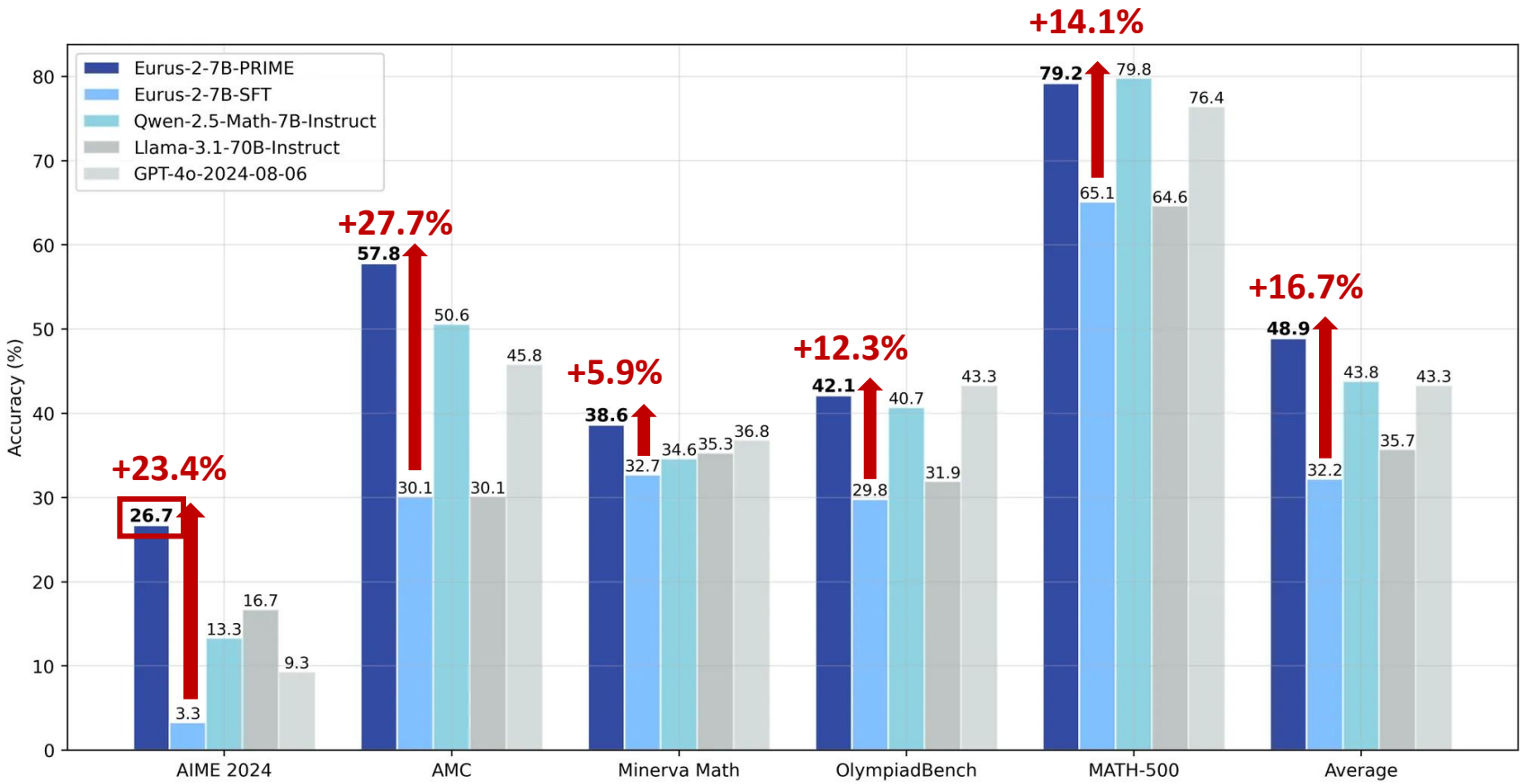


SFT ref: We retrain the initial policy to provide π_{ref} for PRM and KL



A Journey Towards Eurus-2

Eurus-2-7B-PRIME achieves **26.7% pass@1 on AIME 2024**, surpassing **GPT-4o**, Llama3.1-70B, and Qwen2.5-Math-7B-Instruct.



A Journey Towards Eurus-2

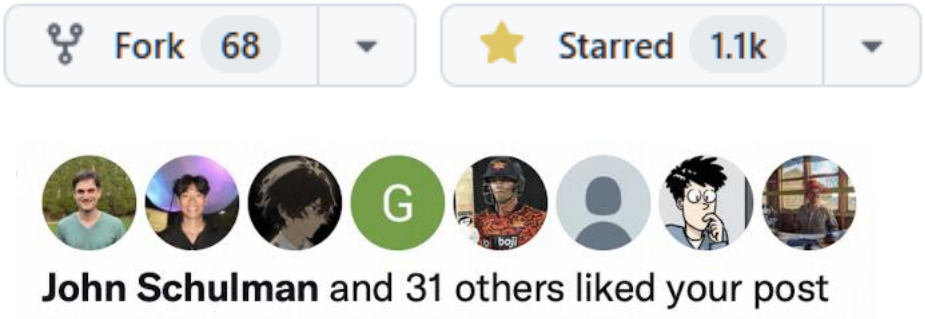
Eurus-2-7B-PRIME needs only 1/10 data of Qwen-Math

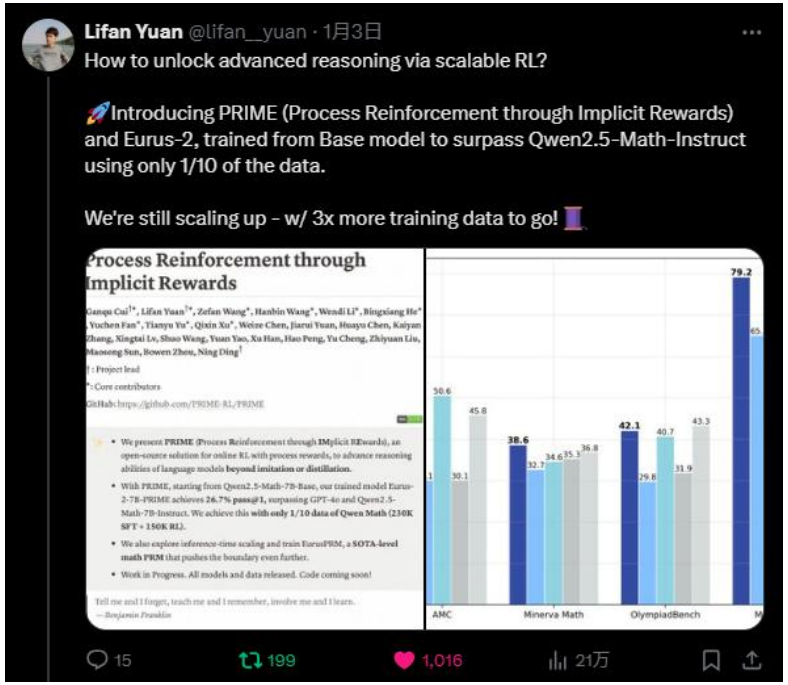
	Eurus-2-7B-PRIME	Qwen2.5-Math-7B-Instruct
Base Model	Qwen2.5-Math-7B	Qwen2.5-Math-7B
SFT Data	**230K (open-source)**	2.5M (open-source and in-house)
RM Data	**0**	618K (in-house)
RM	**Eurus-2-7B-SFT**	Qwen2.5-Math-RM (72B)
RL Data	**150K queries ×4 samples**	66K queries× 32 samples

PRIME: Summary

We find a better way for RL with PRM that

- Could boost model on **math&coding** to be on par with larger models
- Needs no **imitation or distillation**
- With **high sample-efficiency**
- And are **accessible** to every model





Thanks!

Q&A

Paper: https://arxiv.org/abs/2502.01456

Github: https://github.com/PRIME-RL/PRIME

Ganqu Cui

2025.02