SELF-GUIDED PROCESS REWARD OPTIMIZATION WITH REDEFINED STEP-WISE ADVANTAGE FOR PROCESS REINFORCEMENT LEARNING

Wu Fei[1,2] **Hao Kong**[1,*] **Shuxian Liang**[1] **Yang Lin**[1]
Yibo Yang[4] **Jing Tang**[2,3] **Lei Chen**[2,3] **Xiansheng Hua**[1]

[1]Terminus Group
[2]The Hong Kong University of Science and Technology (Guangzhou)
[3]The Hong Kong University of Science and Technology
[4]King Abdullah University of Science and Technology
[*]Corresponding author

ABSTRACT

Process Reinforcement Learning (PRL) has demonstrated considerable potential in enhancing the reasoning capabilities of Large Language Models (LLMs). However, introducing additional process reward models incurs substantial computational overhead, and there is no unified theoretical framework for process-level advantage estimation. To bridge this gap, we propose **S**elf-Guided **P**rocess **R**eward **O**ptimization (**SPRO**), a novel framework that enables process-aware RL through two key innovations: (1) we theoretically demonstrate that process rewards can be derived intrinsically from the policy model itself, and (2) we redefine the step-wise advantage by introducing well-defined Cumulative Process Rewards (**CPR**) and **M**asked **S**tep **A**dvantage (**MSA**), which facilitates rigorous step-wise action advantage estimation within shared-prompt sampling groups. Our experimental results demonstrate that SPRO outperforms vaniila GRPO with 3.4× higher training efficiency and a 17.5% test accuracy improvement. Furthermore, SPRO maintains a stable and elevated policy entropy throughout training while reducing the average response length by approximately $1/3$, evidencing sufficient exploration and prevention of reward hacking. Notably, SPRO incurs no additional computational overhead compared to outcome-supervised RL methods such as GRPO, which benefit industrial implementation.

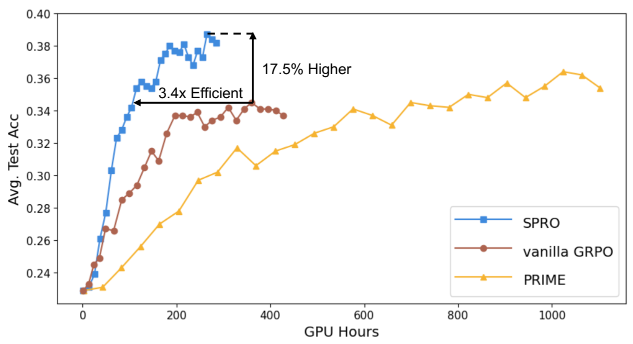
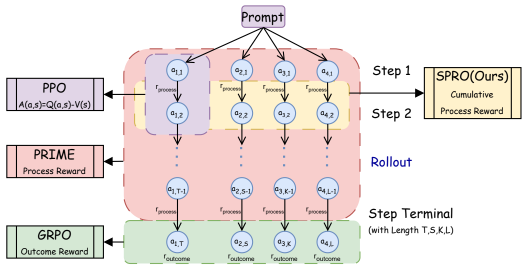
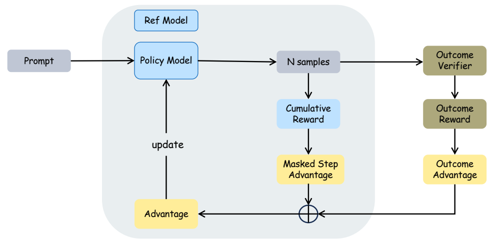


Figure 1: **Performance comparison on math and code benchmarks up to 432 steps.** SPRO outperforms outcome-supervised GRPO with 3.4× higher training efficiency and a 17.5% test accuracy improvement. Notably, SPRO reduces per-step computation time owing to its shorter trajectories.

[1]This work was conducted during the first author's internship at TERMINUS Group. Please contact us at `fwu750@connect.hkust-gz.edu.cn`, `kong.hao@tslsmart.com`, or `konghao@pku.edu.cn`.



(a) **The directions of advantage calculation across different framework.**



(b) **Illustration of SPRO.**

Figure 2: (a): GRPO treats all terminal states as one-step transitions from the initial prompt state. PRIME aggregates all process rewards into a single group for normalization. PPO estimates advantages based on single-step state transitions. In contrast, we propose SPRO, which groups rewards at the same step for calculation. (b): SPRO utilizes Cumulative Process Rewards directly derived from the policy model, thereby eliminating the need for an additional process reward model and establishing a dual-model framework comparable to outcome-supervised approaches.

1 Introduction

Reasoning capacity constitutes a fundamental component of language models' general intelligence, playing a pivotal role in advancing artificial general intelligence (AGI). Recent open source reasoning models, such as OpenAI's o1 [1] and DeepSeek's R1 [2], demonstrate the effectiveness of reinforcement learning in reasoning tasks. Most current RL algorithms [2, 3, 4, 5] optimize policy models based exclusively on outcome rewards while neglecting intermediate feedback. This sparse reward paradigm leads to inefficient learning [6, 7], highlighting the need for efficient and scalable process-based reinforcement learning algorithms. This naturally leads to a fundamental research question: *how to provide well-defined process rewards and guide the policy optimization effectively?*

Although significant research [8, 9, 10, 11] has been devoted to addressing this question, there is no unified theoretical framework for process-level advantage estimation. Training auxiliary process reward models (PRMs), which estimate the future success of intermediate steps, is a commonly adopted strategy. However, PRMs exhibit several widely recognized practical limitations:
(1) Difficult to train: Human-annotated process-level labels lack scalability [8], while automatic annotation often fails to provide reliable supervision [2].
(2) High computational cost: In contrast to the widely adopted dual-model framework (policy and reference model) in outcome-supervised algorithms such as [2, 3, 5], which significantly contributes to scalability and industrial adoption, PRM-based methods introduce an auxiliary reward model. Loading of the additional model requires considerable GPU memory allocation, which constrains the batch size and substantially degrades training throughput and efficiency.
(3) Non-scalable utilization: Existing methods typically leverage PRMs to rerank candidate responses [12] or perform Monte Carlo Tree Search (MCTS) [8, 10, 11], aiming to improve reasoning trajectories. However, constructing the reasoning search space requires sequential rollouts at each step [9], making these approaches non-scalable in online RL.

Recent works have proposed novel methods for acquiring high-quality PRM. Rafailov et al. [13] demonstrate that a well-trained DPO model can inherently achieve credit assignment, effectively expressing token-level rewards in the Markov Decision Process (MDP) framework of LLMs. Extending the DPO framework, Yuan et al. [14] introduce a more generalized implicit PRM training paradigm that replaces preference pairs with point-wise labeled trajectories, which can be trained using standard cross-entropy loss. Subsequently, Cui et al. [15] improve the implicit PRM methodology by proposing the PRIME framework, which effectively combines token-level rewards with outcome rewards to calculate trajectory advantages. The rigorously derived token-level reward functions proposed in [13, 14, 15] eliminate the need for explicit process annotations, thus streamlining the training pipeline. The annotation-free token-level reward addresses the practical limitation (1) mentioned earlier in training PRMs, making this methodology widely adopted in subsequent research.

However, PRIME relies on an auxiliary reward model π_φ to parameterize implicit PRM, which requires iterative training updates throughout the optimization process. This approach not only consumes additional GPU memory but also introduces non-negligible computational overhead. Additionally, after deriving token-level dense rewards, PRIME estimates advantages via a Monte Carlo estimator combined with a leave-one-out baseline. Notably, since each token decoding have distinct states under token-level MDPs [13, 15], the implementation that aggregating all process rewards into a single group for normalization (refer to Fig. 2a) deviates from the advantage-based policy gradient method such as PPO, which will introduce a significant estimation bias.

To address the computational inefficiency of auxiliary process reward models in industrial process reinforcement learning while achieving a less biased process advantages estimation, in this paper, we propose Self-guided Process Reward Optimization (SPRO), a PRM-free algorithm for process reinforcement learning as shown in Fig. 2b. We demonstrate that process rewards can be self-guided directly from the policy model itself, as SPRO eliminates both the annotation requirements and computational overhead inherent to PRM-based approaches and preserves the simplicity and scalability of outcome-supervised RL algorithms [3, 5], which benefit the industrial implementation.

Moreover, SPRO offers a theoretical framework for step-level advantage estimation by redefining the step-wise advantage through a novel Cumulative Process Reward (CPR). This approach aligns with the traditional advantage-based policy gradient framework, while leveraging the capability of encoder-based mask attention to structurally encode prefix-sequence information. Specifically, CPR implicitly aggregates the process rewards from all preceding steps in the prefix sequence as a surrogate for process rewards, enabling more accurate expected return estimation at each timestep. For advantage estimation, we extend the formulation of group-relative advantage from outcome-supervised algorithms [3, 5] and investigate the estimation of step-level advantage. To enable fair comparisons, we introduce Masked Step Advantage (MSA), which enforces strict per-step comparison within shared-prompt sampling groups.

As shown in Fig. 2a, we compare the computation of advantage functions across mainstream methods. Assume that four responses are sampled, where each response receives a outcome reward, and each intermediate step is assigned a process reward. GRPO [2] estimates policy gradients by computing relative advantages within *trajectory groups* using outcome rewards, while PRIME generalizes the grouping paradigm across all prefix sequences as a unified group for normalization. As for our SPRO, we employ the Cumulative Process Reward (CPR) to compute the step-wise rewards from the initial state to each time step t. Subsequently, for each identical step across different trajectories, we perform group-wise normalization of these rewards to obtain the Masked Step Advantage (MSA).

The experimental results demonstrate significant improvements of SPRO over baseline methods. As shown in Fig. 1, our SPRO achieves 17.5% higher test accuracy than vanilla GRPO and 8.3% higher than PRIME, while reducing computational costs to 29% (vs. GRPO) and 15% (vs. PRIME) of GPU hours for equivalent performance. The comparisons on response length and policy entropy also demonstrate that our approach simultaneously addresses two long-standing challenges that have attracted significant community attention: (1) improving token efficiency in reasoning [6, 16], and (2) mitigating policy entropy collapse (or reward hacking) during RL training [7]. We discuss these phenomena in Sec. 5. This dual improvement indicates that our framework enables the policy model to better be aware of the advantages of each step, resulting in both computationally efficient reasoning and more effective action space exploration.

The main contributions are summarized as follows:

- We introduce a novel RL framework for LLMs *Self-Guided Process Reward Optimization (SPRO)*, which eliminates the need for expensive Process Reward Models and retains the same simplicity and scalability as outcome-supervised RL.
- We redefine the step-level advantage by introducing a novel *Cumulative Process Reward (CPR)* as a surrogate for self-guided process rewards and further propose *Masked Step Advantage (MSA)*, which enables a strict per-step comparison within shared-prompt sampling groups to estimate step-level advantages.
- Our experimental results demonstrate that SPRO simultaneously improves accuracy and training efficiency while resolving two critical challenges: token efficiency and policy entropy collapse. SPRO significantly reduces the length of reasoning sequences while achieving higher accuracy. Moreover, SPRO maintains higher policy entropy, promoting more efficient exploration and mitigating reward hacking.

2 Preliminaries

In this section, we first introduce the token-level MDP for large language models, along with some definitions of reward and objective function in reinforcement learning.

2.1 Token-level MDP for Large Language Models

Following with [14, 15, 13], the token-level MDP is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, f, r, \rho)$.

- The action space $\mathcal{A}$ consists of the combined input and output vocabulary of any given large language model, while the state space $\mathcal{S}$ comprises all input-output states during the inference process. For example, state $\mathbf{s}_t$ at timestep t could be represented as $\mathbf{s}_t = (\mathbf{x}, \mathbf{y}_{<t})$, where $\mathbf{x}$ is the initial input (prompt) and $\mathbf{y}_{<t}$ is the sequence of tokens generated up to step $t-1$.
- $f(\mathbf{s}_t, \mathbf{a}_t)$ represents a state transition model that updates the state $\mathbf{s}_{t+1}$ by concatenating the newly generated token $\mathbf{a}_t$ to $\mathbf{s}_t$. Formally, this can be expressed as $\mathbf{s}_{t+1} = f(\mathbf{s}_t, a_t)$.
- $\rho(\mathbf{s}_t)$ represents a state distribution constraint that limits the sampling range for each state $\mathbf{s}_t$.
- $r(\mathbf{s}_t, \mathbf{a}_t)$ denotes the token-level reward given after the model outputs token $\mathbf{a}_t$ with input state $\mathbf{s}_t$.

2.2 Maximum Entropy Reinforcement Learning in the Token-level MDP

Given a well-defined token-level MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, f, r, \rho)$, we can optimize the process reinforcement policy π_θ by using the following entropy-augmented [17, 18], KL-constrained objective [19, 13]:

$$\max_{\pi_\theta} \mathbb{E}_{\substack{\mathbf{s}_0 \sim \rho(\mathbf{s}_0), \\ \mathbf{a}_t \sim \pi_\theta(\cdot|\mathbf{s}_t)}} \left[\sum_{t=0}^{T} \left(\underbrace{(r(\mathbf{s}_t, \mathbf{a}_t)}_{\text{token reward}} + \beta \underbrace{\log \pi_{\text{ref}}(\mathbf{a}_t|\mathbf{s}_t)}_{\text{KL penalty}} \right) + \beta \underbrace{\mathcal{H}(\pi_\theta)}_{\text{entropy}} \right]. \tag{1}$$

As mentioned in [13, 18], in the general maximum entropy RL setting, the fixed point solution of Eq. (1) is given as:

$$\pi^*(\mathbf{a}_t|\mathbf{s}_t) = e^{(Q^*(\mathbf{s}_t, \mathbf{a}_t) - V^*(\mathbf{s}_t))/\beta}, \tag{2}$$

where $\pi^*(\mathbf{a}|\mathbf{s})$ is the optimal policy and $Q^*(\mathbf{s}, \mathbf{a})$ is the corresponding optimal soft Q-function. The optimal value function V^* is defined as:

$$V^*(\mathbf{s}_t) = \beta \log \sum_{\mathbf{a} \in \mathcal{A}} e^{Q^*(\mathbf{s}_t, \mathbf{a})/\beta}. \tag{3}$$

As shown in Eq. (2), the relationship between the reward function $r(\mathbf{s}, \mathbf{a})$ and optimal policy $\pi(\mathbf{a}|\mathbf{s})$ is not a direct mapping. Instead, the policy is expressed through Q-function and V-function, which themselves represent estimates of total future returns.

To further investigate the reward-policy relationship, Rafailov et. al. [13] introduced a modified equality between reward function and value functions using KL-divergence penalty, where:

$$Q^*(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} r(\mathbf{s}_t, \mathbf{a}_t) + \beta \log \pi_{\text{ref}}(\mathbf{a}_t|\mathbf{s}_t) + V^*(\mathbf{s}_{t+1}), & \text{if } \mathbf{s}_{t+1} \text{ is not terminal,} \\ r(\mathbf{s}_t, \mathbf{a}_t) + \beta \log \pi_{\text{ref}}(\mathbf{a}_t|\mathbf{s}_t), & \text{if } \mathbf{s}_{t+1} \text{ is terminal.} \end{cases} \tag{4}$$

It should be noted that some prior work [20, 21] has also proposed similar definitions, but they required an assumption that the discount factor $\gamma < 1$. Rafailov et. al. [13] further proved that the relationship in Eq. (4) is indeed one-to-one in the token MDP under mild assumptions, which means there is a **bijection** between reward functions and corresponding optimal Q-functions in the token-level MDP.

By log-linearizing the optimal policy fixed point in Eq. (2)

$$\beta \log \pi^*(\mathbf{a}_t|\mathbf{s}_t) = Q^*(\mathbf{s}_t, \mathbf{a}_t) - V^*(\mathbf{s}_t). \tag{5}$$

Substituting in the Bellman equation from Eq. (4) [22, 23], we have the following function:

$$r(\mathbf{s}_t, \mathbf{a}_t) + V^*(\mathbf{s}_{t+1}) - V^*(\mathbf{s}_t) = \beta \log \frac{\pi^*(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\text{ref}}(\mathbf{a}_t|\mathbf{s}_t)}. \tag{6}$$

This establishes a mathematical relationship between the process reward function and the policy model.

3 Self-Guided Process Reward Optimization

In this section, we propose a novel PRM-free process reinforcement learning framework for token-level Markov Decision Processes (MDPs) in LLMs. Our framework uniquely enables the policy model to serve dual roles during optimization: (i) as an **Actor** module for policy improvement through reinforcement learning, and (ii) as a **Reward** module for token-level credit assignment during the generation process. Therefore, we refer to our framework as Self-Guided Process Reward Optimization.

We introduce our framework through three steps:

- Sec. 3.1: We propose the hypothesis that any LLM can provide credit assignment for token-level MDP, which is independent of the specific training objective used to train the LLM.
- Sec. 3.2: We redefine step-wise advantage by introducing Cumulative Process Reward (CPR) for token-level MDP and further propose Masked Step Advantage for process RL training.
- Sec. 3.3: We introduce the Self-Guided Process Reward Optimization (SPRO) algorithm, detailing its objective function and training procedure.

This self-guided framework has three key advantages: (i) it avoids the reward modeling bottleneck inherent in traditional RLHF pipelines; (ii) credit assignment dynamically improves in alignment with policy improvement, creating a virtuous cycle of mutual refinement; and (iii) it enables effective process reinforcement learning.

3.1 The process rewards could be self-guided

As shown in Eq.(6), the process reward $r(\mathbf{s}_t, \mathbf{a}_t)$ is defined as the log-ratio between the probability of $\mathbf{a}_t$ under the optimal policy and the given reference policy. Rafailov et al.[13] argue that the trained DPO model π^* yields the best estimate of an optimal Q-function, since the value term $V^*(\mathbf{s})$ is reduced using Bradley-Terry preference model. Building upon this, Yuan et al.[14] further extend the idea to Cross-Entropy (CE) loss. Cui et al.[15] apply CE loss to train an implicit PRM, and further use the resulting process rewards to compute advantages to update the policy model.

Obviously, the accuracy of $r(\mathbf{s}_t, \mathbf{a}_t)$ directly depends on the quality of the optimal policy. Since the policy model is trained to directly approximate π^*, the near-optimal solution π_{θ_T} inherently provides more accurate rewards than a PRM π_φ trained separately. Otherwise, the policy model itself would be inferior, contradicting its optimality hypothesis, which means that the separately trained PRM could be a better solution than our trained policy model π_θ. This observation forms the theoretical foundation for our self-guided reward formulation.

Proposition 1. *Any LLM is always the optimal soft Q-functions for some reward functions in the token-level MDP [13], thus enabling token-level credit assignment. In particular, LLMs with stronger downstream task performance provide more accurate credit assignment.*

Proof. Let $\boldsymbol{\ell}(\mathbf{a}_t|\mathbf{s}_t)$ denote the output logits of a given LLM policy π for token $\mathbf{a}_t$ conditioned on state $\mathbf{s}_t$. We define Q-function as a scaled version of the logits: $Q(\mathbf{s}_t, \mathbf{a}_t) = \beta\boldsymbol{\ell}(\mathbf{a}_t|\mathbf{s}_t)$. The corresponding partition function is derived by taking the log-sum-exp of logits over all possible actions $\mathbf{a} \in \mathcal{A}$ and defined as $Z(\mathbf{s}_t)$. Consequently, the optimal value function corresponding to such Q is exactly $V(\mathbf{s}_t) = \beta \log Z(\mathbf{s}_t)$. This yields the following form of the policy:

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \operatorname{softmax}(\boldsymbol{\ell}(\mathbf{a_t}|\mathbf{s_t})) = \frac{e^{\boldsymbol{\ell}(\mathbf{a_t}|\mathbf{s_t})}}{\sum_{\mathbf{a}\in\mathcal{A}} e^{\boldsymbol{\ell}(\mathbf{a}|\mathbf{s_t})}} = \frac{e^{Q(\mathbf{s}_t,\mathbf{a}_t)/\beta}}{Z(\mathbf{s_t})} = e^{(Q(\mathbf{s}_t,\mathbf{a}_t)-V(\mathbf{s}_t))/\beta}. \tag{7}$$

Eq. (7) shows that any LLM is a soft Q-function for some reward function [13]. Since downstream task performance provides an explicit indicator of how closely an LLM approximates the optimal policy, combined with Eq. (2) and Eq. (6), it follows that the quality of corresponding token-level credit assignment is also observable. Importantly, this property is independent of the specific training objective used to train the LLM. □

In our framework, we utilize log-probabilities extracted from the policy model itself, rather than use a well-trained reward model. Since both the policy and reference models are initialized from the same SFT model, the process rewards are initially zero. As training progresses and the policy model shifts away from the reference, the process rewards start contributing to the RL optimization.

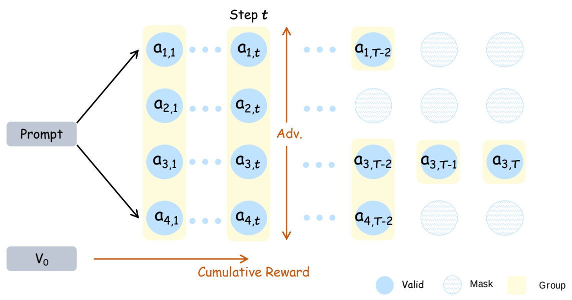


Figure 3: **Illustration of Masked Step Advantage.** Assume that four responses are sampled for each prompt. At each step t, we calculate cumulative rewards and further compute the step-level advantages within the vertical valid masked groups, excluding empty step units from all calculations.

3.2 Redefine Step-wise Advantage

In the previous section, we propose that the process reward can be self-guided by the policy model. In this section, we redefine step-wise advantage by introducing *Cumulative Process Reward (CPR)* and *Masked Step Advantage (MSA)*.

Cumulative Process Reward (CPR). We argue that the mechanism of LLMs should inform the design of process rewards in token-level MDPs. Since auto-regressive generation employs masked attention, the hidden state at step t inherently encodes all information of the prefix sequence [24], which means each hidden state represents the complete trajectory up to its corresponding time step. Prior work has effectively utilized this property: Lightman et al. [8] employ the final token's hidden state at each step to predict correctness in process reward models, aligning with probing studies that leverage such representations to analyze model properties [25, 26, 27, 28, 29, 30]. Therefore, we propose that intermediate reward signals at step t should similarly capture contributions from all preceding steps, which we formalize as the definition of the Cumulative Process Reward (CPR).

Given a policy model π_θ during training iterations (we omit the iteration subscript for convenience), Proposition 1 establishes that there always exists an implict reward function $r(\mathbf{s}_t, \mathbf{a}_t)$ together with corresponding $Q(\mathbf{s}_t, \mathbf{a}_t)$ and $V(\mathbf{s}_t)$ functions. By virtue of the optimality of the Q-function, these functions also satisfy the equality relation in Eq.(6) as follows:

$$r(\mathbf{s}_t, \mathbf{a}_t) + V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t) = \beta \log \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\text{ref}}(\mathbf{a}_t|\mathbf{s}_t)}. \tag{8}$$

For an arbitrary time step t within a trajectory $\tau = \{\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{a}_{T-1}, \mathbf{s}_T\}$, we define a cumulative reward w.r.t. step t by accumulating Eq. (8) from 0 to t:

$$\sum_{j=0}^{t} \left(r\left(\mathbf{s}_j, \mathbf{a}_j\right) + V\left(\mathbf{s}_{j+1}\right) - V\left(\mathbf{s}_j\right)\right) = \sum_{j=0}^{t} \beta \log \frac{\pi_\theta(\mathbf{a}_j|\mathbf{s}_j)}{\pi_{\text{ref}}(\mathbf{a}_j|\mathbf{s}_j)}. \tag{9}$$

By telescoping the value function $V(\mathbf{s}_{j+1}) - V(\mathbf{s}_j)$ on the left-hand side, we can get the following:

$$\sum_{j=0}^{t} r(\mathbf{s}_j, \mathbf{a}_j) + V(\mathbf{s}_{t+1}) = V(\mathbf{s}_0) + \sum_{j=0}^{t} \beta \log \frac{\pi(\mathbf{a}_j|\mathbf{s}_j)}{\pi_{\text{ref}}(\mathbf{a}_j|\mathbf{s}_j)}. \tag{10}$$

The left-hand side of Eq. (10) represents the cumulative reward up to step t plus the future expected return starting from $\mathbf{s}_{t+1}$ (discount factor $\gamma = 1$). Let $\mathcal{R}_t$ denote Cumulative Process Reward (CPR):

$$\mathcal{R}_t := \sum_{j=0}^{t} r(\mathbf{s}_j, \mathbf{a}_j) + V(\mathbf{s}_{t+1}) = V(\mathbf{s}_0) + \sum_{j=0}^{t} \beta \log \frac{\pi(\mathbf{a}_j|\mathbf{s}_j)}{\pi_{\text{ref}}(\mathbf{a}_j|\mathbf{s}_j)}. \tag{11}$$

We adopt CPR to align with the cumulative nature of LLM representations. Additionally, since all responses start from the same initial state $\mathbf{s}_0$, $\mathcal{R}_t$ facilitates subsequent advantage computation.

Algorithm 1: **S**elf-guided **P**rocess **R**eward **O**ptimization (**SPRO**)

```
Input: Initial policy model $\pi_{\theta_{\text{init}}}$; outcome reward verifier $r_o$; task prompts $\mathcal{D}$.
1  policy model $\pi_\theta \leftarrow \pi_{\theta_{\text{init}}}$
2  reference model $\pi_{\text{ref}} \leftarrow \pi_{\theta_{\text{init}}}$
3  for iteration $= 1$ to $K$ do
4      Sample a batch $\mathcal{D}_b$ from $\mathcal{D}$
5      Update the old policy model $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$
6      Sample G outputs $\{\mathbf{y}_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot \mid \mathbf{x})$ for each $x \in \mathcal{D}_b$
7      Compute outcome rewards $\{r_o(\mathbf{y}_i)\}_{i=1}^G$ for each sampled output $\mathbf{y}_i$
8      Apply accuracy filter on prompts in $\mathcal{D}_b$
9      Compute the Cumulative Process Reward $\mathcal{R}_{i,t}$ and Masked Step Advantage $\text{MSA}_{i,t}$ for the
         $t$th token of all responses $\{\mathbf{y}_i\}$ with Eqs. (11-12)
10     Compute $A_{i,t}$ for the $t$th token of all responses $\{\mathbf{y}_i\}$ through Eq.(13)
11     for iteration $= 1$ to $\mu$ do
12         Update the policy model $\pi_\theta$ by maximizing the SPRO objective Eq.(14)
13     end
14 end
Output: Policy Model $\pi_\theta$.
```

Masked Step Advantage (MSA). For trajectories $\{\tau_i\}$ of the same prompt, the Cumulative Process Rewards $\{\mathcal{R}_{i,t}\}$ at the same step t are comparable because they all start from the same initial state $\mathbf{s}_0$ and can be regarded as one-step state transition rewards, similar to GRPO that the outcome reward can be seemed as one-step transition rewards. We formally define **Masked Step Advantage (MSA)** corresponding to the cumulative reward as follows:

$$\text{MSA}_{i,t} := \mathcal{R}_{i,t} - b_t = \tilde{\mathcal{R}}_{i,t} - \tilde{b_t} = \tilde{\mathcal{R}}_{i,t} - \text{mask\_mean}(\{\tilde{\mathcal{R}}_{i,t}\}), \tag{12}$$

where i represents the ith response and $\tilde{\mathcal{R}}_{i,t} = \sum_{j=0}^{t} \beta \log \frac{\pi_\theta(\mathbf{a}_j|\mathbf{s}_j)}{\pi_{\text{ref}}(\mathbf{a}_j|\mathbf{s}_j)}$. The term b_t (or $\tilde{b}_t$) denotes the vertical group-wise average of valid scores at step t, serving as an advantage baseline. Here, we compute them using a masked_mean operator. Note that the constant $V(\mathbf{s}_0)$ on the right-hand side of Eq. (11) will cancel out in all responses, making $\mathcal{R}_{i,t}$ and $\tilde{\mathcal{R}}_{i,t}$ equivalent for advantage calculation.

Taking Fig. 3 as an example, if only the 3th response contains a valid step at time $T-1$, then $b_{T-1} = \mathcal{R}_{3,T-1}$ and $\text{MSA}_{3,T-1} = 0$. This indicates that **MSA does not introduce the length bias**, since the third response does not gain additional advantage even if it is longer than the others. In this way, strict per-step comparisons within shared-prompt sampling groups are achieved without introducing length bias.

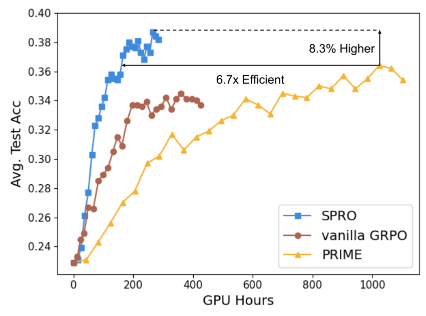
3.3 Self-Guided Process Reward Optimization

Following common practice in Policy Gradient algorithm [31], we incorporate MSA into the outcome-supervised RL method Group Relative Policy Optimization (GRPO) [3], resulting in the SPRO advantage function:

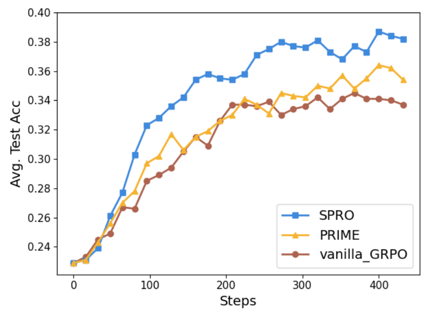
$$A_{i,t} \quad = \quad \underbrace{\frac{r_o(\mathbf{y}_i) - \text{mean}(\{r_o(\mathbf{y}_i)\})}{\text{std}(\{r_o(\mathbf{y}_i)\})}}_{\text{GRPO with outcome rewards}} + \underbrace{\Big(\mathcal{R}_{i,t} - \text{masked\_mean}(\{\mathcal{R}_{i,t}\})\Big)}_{\text{MSA}_{i,t}}, \tag{13}$$

Then the policy model can be optimized by maximizing the objective as follows:

$$\begin{aligned}\mathcal{J}_{\text{SPRO}}(\theta) \quad &= \mathbb{E}_{\mathbf{x},\{\mathbf{y}_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|\mathbf{x})} \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|\mathbf{y}_i|} \sum_{t=1}^{|\mathbf{y}_i|} \min \\ &\left(\frac{\pi_\theta(y_{i,t} \mid \mathbf{x}, \mathbf{y}_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t} \mid \mathbf{x}, \mathbf{y}_{i,<t})} A_{i,t}, \text{CLIP}\left(\frac{\pi_\theta(y_{i,t} \mid \mathbf{x}, \mathbf{y}_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t} \mid \mathbf{x}, \mathbf{y}_{i,<t})}, 1-\varepsilon, 1+\varepsilon \right) A_{i,t} \right)\end{aligned} \tag{14}$$



(a) GPU Hours



(b) Steps

Figure 4: Performance of SPRO on math and code benchmarks with respect to GPU hours and training steps. SPRO outperforms the previous SoTA process reinforcement learning method PRIME with 6.7× higher training efficiency and an 8.3% performance improvement. Notably, SPRO reduces per-step computation time owing to its shorter trajectories.

Table 1: **Comparison of test accuracy** between SPRO and other baselines using the same base model and training data. We present pass@1 accuracy scores at step 400 on four math benchmarks and two code benchmarks.

Methods	AMC	MATH	Olympiad	K12	CodeForces	CodeContests	Avg.
Base Model	21.1	48.1	13.9	40.0	5.2	11.3	22.8
vanilla GRPO	23.6	51.8	20.5	48.6	28.6	28.1	33.5
PRIME	31.2	52.7	25.4	54.6	26.4	25.9	36.0
Ours (SPRO)	**31.9**	**53.6**	**28.2**	**55.0**	**29.4**	**32.1**	**38.4**

Alg. 1 illustrates the detailed implementation of our proposed SPRO framework. It can be observed that the calculations of cumulative rewards and masked step advantages depend exclusively on the current policy model during training, which motivates our designation of the approach as self-guided. Furthermore, our proposed advantage function computes relative advantages by grouping tokens from the identical timestep across all sampled responses, ensuring a less biased advantage estimation.

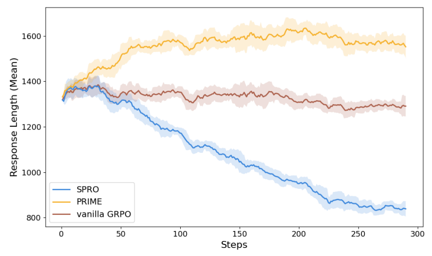
4 Experiments

4.1 Experimental setup

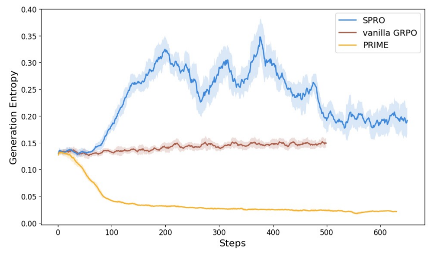
Base model and Dataset. We adopt Eurus-2-7B-SFT [15] as our base model, which is fine-tuned from Qwen2.5-Math-7B-Base [32] on mathematical and programming tasks. The RL dataset is Eurus-2-RL-Data [15], which contains math problems ranging from the high school level to International Mathematical Olympiad competition questions, as well as programming tasks, primarily at the competitive programming level. We evaluate models on AMC [33], MATH-500 [34], Olympiad-Bench [35], CodeForces [36], and CodeContests [37].

Resources and Hyper-parameters. All experiments are conducted on a single node equipped with 8x NVIDIA A800 GPUs (80G memory) using the veRL framework [38]. For optimization, we use the AdamW optimizer with a cosine decay learning rate schedule, initialized at 1×10^{-6}. For rollout stage, we collect 256 prompts with an oversampling factor of 2 and generate 4 responses per prompt. We apply an accuracy filtering threshold between 0.2 and 0.8, and prompts that fall within this range are prioritized. For training, the batch size is 256 and the micro batch size is 16. The KL coefficient is set to 0, and the entropy coefficient is set to 0.001 for all experiments.

Outcome Verifiers. We employ rule-based reward modeling for outcome verification, aligning with common practices in the recent literature [2, 15, 39]. Specifically, we assign a score of 1 if a math answer matches the ground truth and 0 otherwise. For code generation task, we compute the reward as the proportion of passed test cases.



(a) Average length of policy model-generated responses.



(b) Entropy of policy model's generation probabilities.

Figure 5: Effective process reinforcement learning enables efficient reasoning traces and exploration of action space.

Baseline Algorithms. We use vanilla GRPO [3] as the baseline algorithm for outcome-supervised RL training, and the previous SoTA PRIME [15] as the baseline algorithm for process-supervised RL training. We maintain consistency in the base model, training dataset, and key hyperparameters with those employed in our proposed method.

4.2 Main Results

Experiments on mathematical and programming tasks demonstrate that SPRO effectively enables process RL for reasoning, yielding substantial improvements over GRPO and PRIME.

Accuracy Improvement As shown in Fig. 1 and Fig. 4, SPRO achieves a 17.5% higher test accuracy than vanilla GRPO [3], and 8.3% higher than PRIME [15]. The comparative results in Table. 1 further demonstrate that our SPRO achieves significant performance advantages in enhancing the reasoning capabilities of policy models.

Training Efficiency. As shown in Fig. 1 and Fig. 4a, our SPRO requires only 29% and 15% of the GPU hours needed by vanilla GRPO and PRIME respectively to achieve equivalent accuracy. Furthermore, the shorter sampling sequence length generated by SPRO (as illustrated in Fig. 5a) contributes to a significant computational advantage in each optimization step. This is evidenced by comparing the computation time required for the same number of training steps across different methods (see comparative results in Fig. 4).

Entropy Stability. As shown in Fig. 5b, unlike PRIME which suffers from entropy collapse, our method maintains effective state-action space exploration during training, preserving optimization efficiency and avoiding reward hacking.

5 Analysis

5.1 SPRO enables efficient reasoning trajectories

As proposed by Qu et al. [6], "efficiency is the essence of intelligence". Efficient reasoning is essential for training, inference, and real-world deployment. However, existing RL training algorithms struggle to achieve such token-efficient behaviors. Fig. 5a shows that SPRO reduces the average response length of vanilla GRPO [3] by nearly one-third while improving test accuracy by 17.5%, demonstrating the effectiveness of our framework in process reinforcement learning.

This improvement is primarily attributed to the rigorous step-wise comparison mechanism introduced in Eq. (12). Our approach provides the policy model with MSA feedback at each generation step, making it aware which tokens contribute positively to the overall return. Such a fine-grained feedback mechanism effectively encourages more concise and task-focused output.

In contrast, existing methods such as PRIME [15] employ a coarser reward signal by averaging the returns across all trajectories and timesteps. This design results in an advantage function that is relative to both groups and timesteps, consequently diminishing the effectiveness of the intended group-level comparison.

The Cumulative Process Rewards defined in Eq. (11) not only provide the theoretical foundation for MSA in Eq. (12), but also exhibit intrinsic alignment with the hidden state dynamics of LLMs. This insight, inherent in LLMs mechanisms and different from conventional RL scenarios, is deserving of wider attention in future developments of LLM reinforcement learning.

5.2 SPRO ENABLES EXPLORATION OF ACTION SPACE

The collapse of policy entropy is a widely observed phenomenon in reinforcement learning, as extensively documented in prior works [7, 40], and our experimental results are consistent with this trend. In our experiments, the entropy coefficient is fixed at 0.001 across methods. As shown in Fig. 5, the policy entropy starts at 0.13 due to SFT initialization. PRIME [15] suffers a sharp entropy drop within the first 100 steps, while GRPO [3] remains largely stable. In contrast, our SPRO demonstrates active state action space exploration, whose policy entropy increases to 0.35 and remains relatively high up to 500 steps before gradually declining and fluctuating within a narrow range around 0.2.

This persistent exploration constitutes a key feature of SPRO, enabling longer and more effective training. By maintaining output diversity, our SPRO prevents premature convergence to suboptimal behaviors while preserving the potential for further improvement. Cui et al. [7] demonstrate that the policy entropy naturally decreases when high-advantage actions already have high probability, but increases when the model selects rare yet high-advantage actions. SPRO encourages the latter behavior, validating the effectiveness of our advantage function design in Eq. 13. This exploration mechanism directly contributes to the 17.5% improvement in test accuracy over vanilla GRPO. Importantly, this performance gain stems not from implementation tricks such as policy loss clipping, but from genuine exploration dynamics.

In particular, our SPRO successfully combines active exploration with more concise reasoning trajectories. The reduced response length does not mean shortcutting; instead, the policy model thoroughly explores the state-action space while strategically selecting concise, diverse, and effective solutions. This demonstrates a form of intelligent exploration, where the model identifies efficient solutions without compromising the correctness or diversity.

5.3 SPRO ENABLES INDUSTRY-SCALE PROCESS REINFORCEMENT LEARNING

Even in advanced industrial Large Reasoning Models, PRM has been identified as a failure case due to its inherent limitations [2]. Training inefficiency significantly reduces its potential benefits. This is clearly demonstrated in Fig. 4 when the x-axis is changed from training steps to GPU hours. Taking PRIME [15] as an illustrative example, loading the additional PRM model consumes significant GPU memory, which constrains the micro-batch size for the forward and backward propagation of the policy model. Moreover, updating the PRM introduces non-negligible computational overhead.

As shown in Fig. 4a, our method reduces the additional computational overhead. Due to hardware limitations, our experiments are conducted on 7B models. However, SPRO demonstrates considerable potential for industrial-scale implementation, as its principle of self-guided process rewards capitalizes on the inherent capacity of the policy model, enabling scaling laws to remain effective.

6 CONCLUSION

In this work, we introduced Self-Guided Process Reward Optimization (SPRO), a novel and scalable RL framework for LLMs that eliminates the dependency on costly Process Reward Models while preserving the simplicity of outcome-supervised RL. By introducing the Cumulative Process Reward (CPR) as a surrogate for self-guided process signals and proposing Masked Step Advantage (MSA), our method enables rigorous step-level advantage estimation through shared-prompt comparisons. Experiments demonstrate that SPRO significantly improves both accuracy and training efficiency. In particular, it addresses two critical challenges in RL for LLMs: (1) token efficiency, where SPRO reduces reasoning sequence lengths while achieving higher task accuracy, and (2) policy entropy collapse, where our method maintains higher entropy levels, promoting more efficient exploration and mitigating reward hacking during training. The scalability and ease of deployment of SPRO make it particularly suitable for industrial implementation, offering a practical and effective alternative to traditional process reward approaches.

REFERENCES

[1] OpenAI. Learning to reason with LLMs. `https://openai.com/index/learning-to-reason-with-llms/`, 2024. Accessed: 15 March 2025.

[2] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[3] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[4] Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

[5] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *CoRR*, 2024.

[6] Xiaoye Qu, Yafu Li, Zhaochen Su, Weigao Sun, Jianhao Yan, Dongrui Liu, Ganqu Cui, Daizong Liu, Shuxian Liang, Junxian He, et al. A survey of efficient reasoning for large reasoning models: Language, multimodality, and beyond. *arXiv preprint arXiv:2503.21614*, 2025.

[7] Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

[8] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

[9] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[10] Xidong Feng, Ziyu Wan, Muning Wen, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

[11] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, 2024.

[12] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

[13] Rafael Rafailov, Joey Hejna, Ryan Park, and Chelsea Finn. From r to Q^*: Your language model is secretly a q-function. In *First Conference on Language Modeling*, 2024.

[14] Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. Free process rewards without process labels. *arXiv preprint arXiv:2412.01981*, 2024.

[15] Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

[16] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

[17] Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

[18] Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.

[19] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[20] Divyansh Garg, Shuvam Chakraborty, Chris Cundy, Jiaming Song, and Stefano Ermon. Iq-learn: Inverse soft-q learning for imitation. *Advances in Neural Information Processing Systems*, 34:4028–4039, 2021.

[21] Joey Hejna, Rafael Rafailov, Harshit Sikchi, Chelsea Finn, Scott Niekum, W Bradley Knox, and Dorsa Sadigh. Contrastive preference learning: Learning from human feedback without reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024.

[22] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[23] Joe Watson, Sandy Huang, and Nicolas Heess. Coherent soft imitation learning. *Advances in Neural Information Processing Systems*, 36:14540–14583, 2023.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[25] Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, 2022.

[26] Carlos Aspillaga, Marcelo Mendoza, and Álvaro Soto. Inspecting the concept knowledge graph encoded by modern language models. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2984–3000, 2021.

[27] Alexis Conneau, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2126–2136, 2018.

[28] Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502, 2022.

[29] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, 2021.

[30] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: part 3.1, knowledge storage and extraction. In *Proceedings of the 41st International Conference on Machine Learning*, pages 1067–1077, 2024.

[31] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[32] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

[33] Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13:9, 2024.

[34] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021.

[35] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, 2024.

[36] Shanghaoran Quan, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren, Bofei Gao, Yibo Miao, Yunlong Feng, et al. Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings. *arXiv preprint arXiv:2501.01257*, 2025.

[37] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[38] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297, 2025.

[39] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. T\" ulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

[40] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *CoRR*, 2025.