# UNSUPERVISED PROCESS REWARD MODELS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Process Reward Models (PRMs) are a powerful mechanism for steering large language model reasoning by providing fine-grained, step-level supervision. However, this effectiveness comes at a significant cost: PRMs require expert annotations for every reasoning step, making them costly and difficult to scale. Here, we propose a method for training *unsupervised* PRMs (uPRM) that requires *no human supervision*, neither at the level of step-by-step annotations nor through ground-truth verification of final answers. The key idea behind our approach is to define a scoring function, derived from LLM next-token probabilities, that jointly assesses candidate positions of first erroneous steps across a batch of reasoning trajectories. We demonstrate the effectiveness of uPRM across diverse scenarios: *(i)* uPRM achieves up to 15% absolute accuracy improvements over the LLM-as-a-Judge in identifying first erroneous steps on the ProcessBench dataset; *(ii)* as a verifier for test-time scaling, where uPRM outperforms the majority voting baseline by up to 7%, and *(iii)* when used as a reward signal in reinforcement learning, uPRM enables more robust policy optimization throughout training compared to a supervised PRM trained using ground-truth labels. Overall, our results open a path toward scalable reward modeling for complex reasoning tasks.

## 1 INTRODUCTION

Improvements in the step-by-step reasoning abilities of large language models (LLMs) have become a cornerstone for their recent success in domains such as mathematics and programming (Chen et al., 2025; Li et al., 2025; Wei et al., 2022; Guo et al., 2025). In order to incentivize or steer the reasoning process in LLMs, one needs to evaluate their correctness. The basic approach to achieve this is by computing a single score for the whole reasoning trajectory (*e.g.*, verifying only the final answer of a solution) using Outcome Reward Models (ORMs) (Cobbe et al., 2021; Guo et al., 2025; Du et al., 2025). However, using such sparse and crude feedback, especially for long chains of thought, is extremely ineffective and can lead to false positives, reinforcing incorrect reasoning traces that ultimately result in formally correct answers (Uesato et al., 2022).

In contrast, Process Reward Models (PRMs) (Lightman et al., 2024) were introduced to produce dense step-wise scores that can guide the reasoning process more gradually. Naturally, such finer control over reasoning leads to improved results in both test-time scaling (TTS) (Snell et al., 2025) and reinforcement learning (RL) (Cheng et al., 2025). Despite their overall advantage over ORMs, PRMs have a significant limitation: they require meticulously labeled training data containing step-by-step annotated reasoning trajectories. To address this problem, numerous frameworks have been developed to infer step-level labels from ground truth final answers based on brute-force Monte Carlo estimations (Wang et al., 2024; Luo et al., 2024) or implicit process reward modeling (Yuan et al., 2025; Cui et al., 2025). However, these approaches still rely heavily on the availability of ground truth answers in the data or access to external verifiers, and are often highly computationally demanding, which limits their general applicability.

In this work, we present an approach for training *fully unsupervised* Process Reward Models (uPRMs) that requires neither step-level annotations nor ground-truth verification of final answers. Our key insight is that LLMs, through their next-token probabilities, implicitly encode judgments about the correctness of reasoning steps. Specifically, we construct sequences that interleave reasoning steps with correctness markers, and extract the probabilities an LLM assigns to these markers to define the scoring function that measures how plausible a given error position is. By evaluating multiple trajectories jointly rather than independently, we leverage the in-context learning capabilities of



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

LLMs to obtain more reliable assessments. We then train uPRM to optimize this joint score via RL, effectively distilling the LLM's evaluation capability into a dedicated process reward model.

We demonstrate the effectiveness of our uPRM through a diverse set of experiments:

- We show that uPRM effectively *identifies the positions of first erroneous steps*, achieving up to $15\%$ absolute accuracy improvements over the LLM-as-a-Judge baseline. Remarkably, uPRM achieves the largest gains on the most challenging datasets such as Olympiad-Bench (He et al., 2024) and Omni-Math (Gao et al., 2025).
- In *test-time scaling experiments*, we show that uPRM outperforms the majority voting baseline by up to 6.9% absolute gains when verifying 256 generations of Llama-3.2-1B-Instruct (Grattafiori et al., 2024) and demonstrate performance gains across multiple LLMs.
- We show that uPRM can be used as a *reward signal for RL*. Surprisingly, compared to a supervised PRM trained with ground-truth labels, which is prone to rapid reward hacking, uPRM supports *more robust policy optimization* across training runs. Although it does not fully eliminate reward hacking, we observe that such failures arise less frequently and tend to be less severe, yielding superior final performance across multiple policy models. For example, uPRM yields a $4\%$ accuracy gain for Qwen2.5-Math-1.5B (Yang et al., 2024b) over training with a verifiable outcome reward.

## 2 Related Work

**Process Reward Models from Outcome Labels.** Since manually obtaining granular annotations can be laborious and expensive (Lightman et al., 2024), a variety of approaches have emerged to take advantage of the available outcome labels to obtain process supervision for training PRMs. For example, Math-Shepherd (Wang et al., 2024) proposed an automatic process annotation procedure that assigns a label to each step based on its potential to lead to a correct final answer. Similar automated annotation techniques were proposed in subsequent works (Luo et al., 2024; Chen et al., 2024; Luo et al., 2025; Kazemnejad et al., 2025). Nevertheless, such techniques only complement the labeling corresponding to actual step correctness (Zhang et al., 2025), and require significant computational resources for Monte Carlo rollouts.

An alternative approach is based on implicit process reward modeling, in which a PRM is learned directly from the outcome rewards, without relying on explicitly annotated reasoning steps. In particular, Yuan et al. (2025) and Cui et al. (2025) develop this idea by introducing a special parameterization of an ORM that allows for interpreting its partial responses as Q-values required for deriving implicit process rewards. Other works suggest leveraging ORM outputs to provide step-wise feedback for training PRMs by computing a relative confidence change (Lu et al., 2024), introducing a modified Bradley-Terry objective (Xie et al., 2025), or by adopting buffering probabilities to reduce label noise (Sun et al., 2025).

While these methods reduce the need for step-level annotations, they remain dependent on access to the ground-truth outcome labels either for assessing Monte Carlo rollouts or for training the underlying ORM. In contrast, our approach eliminates this requirement entirely, *training PRMs without any supervision* at either step-level annotations or outcome labels.

**LLM-as-a-Judge paradigm.** Large language models have been employed as automatic evaluators in various complex tasks due to their ability to process diverse data types and provide flexible assessment, eliminating the need for expert annotations. Prominent instances include MT-Bench and Chatbot Arena (Zheng et al., 2023), as well as AlpacaEval (Dubois et al., 2024), which use strong LLMs to perform pairwise comparisons of candidate responses and aggregate win-rates. GPTScore (Fu et al., 2024) uses the generation likelihood of candidate text given an instruction as a quality measure. G-Eval (Liu et al., 2023) prompts an LLM to output discrete scores and uses token probabilities over score tokens to compute a weighted average, yielding more continuous and stable evaluations.

Most existing LLM-as-a-Judge pipelines operate by prompting an LLM to generate an explicit verdict, and then parsing the generated text into a discrete label or score. Viewed through this lens, our method, instead of sampling a judgment, employs raw next-token probabilities to define a scoring function that measures how plausible a given solution is. Furthermore, while prior work primarily

leveraged LLM judges for offline evaluation and model selection, we convert the judge's probabilistic assessment into an optimization objective that provides direct supervision for training PRMs.

**Test-time Scaling with Process Reward Models.** Test-time scaling (TTS) involves allocating additional compute resources to an LLM during inference to enhance task performance (Wei et al., 2022; Snell et al., 2025; Luo et al., 2024; Liu et al., 2025a). This paradigm includes a sampling strategy to generate diverse candidate answers and a method to select the final response, typically using a reward model (Brown et al., 2024). Common sampling strategies include *(i)* Best-of-N (Brown et al., 2024), where N independent answers are generated and scored, and the answer with highest aggregated score is selected, *(ii)* Beam Search (Snell et al., 2025), in which intermediate nodes within each beam are retained or discarded using scores from reward model, and *(iii)* Diverse Verifier Tree Search (DVTS) (Beeching et al., 2024), which constructs multiple, independent beam search trees to increase response diversity. In addition to these approaches, majority voting is a reward-model-free method that selects the most frequent answer.

One major concern with PRMs in TTS is the effective use of the assigned rewards to select the final response. Current selection methods do not achieve performance comparable to the strong pass@N metric, where a single correct answer out of N generations is sufficient, and have led to recent exploration on improving PRMs (Zhao et al., 2025; Zhang et al., 2025; Yin et al., 2025; Yao et al., 2026). In our work, we observe that *uPRM provides guidance for selecting answers and improving performance from multiple generations as we allocate more test-time compute across different LLMs*.

**Reinforcement Learning with Process Reward Models.** RL has been widely adopted to incentivize reasoning abilities in LLMs, particularly to solve mathematical problems (Guo et al., 2025; Liu et al., 2025b). Most popular frameworks assign a sparse outcome reward for the entire response generated by the policy model. A more desirable option would be to introduce dense intermediate rewards into the reasoning process so that learning becomes more effective (Wang et al., 2024; Setlur et al., 2025; Cui et al., 2025). One of the key challenges in applying PRMs to RL is reward hacking, where the policy learns to exploit spurious patterns in the reward model rather than genuinely improving reasoning quality (Gao et al., 2023; Guo et al., 2025). Existing work has focused on algorithmic mitigations, such as min-form credit assignment (Cheng et al., 2025), but reward hacking is generally considered inevitable when relying solely on PRM rewards. In our experiments, we find that uPRM exhibits *better robustness to reward hacking while improving performance* than a supervised PRM trained on the same dataset.

# 3 Background

## 3.1 Supervised Process Reward Models

Let $\tau = (x, y)$ be a solution trajectory consisting of a problem $x$ and a sequence of reasoning steps $y = (y_1, \ldots, y_T)$ tackling it. We use the prefix notation $y_{1:t} = (y_1, \ldots, y_t)$ and write $\tau_{\leq t} \coloneqq (x, y_{1:t})$ for the partial trajectory up to step $t$. A parametrized process reward model $r_\theta(c_t|\tau_{\leq t})$ defines a distribution over step-correctness labels $c_t \in \{0, 1\}$, where $c_t = 1$ indicates that step $y_t$ is correct[1] and $\theta$ refers to trainable parameters.

In practice, training a PRM requires a labeled dataset $\mathcal{D}$ where each solution trajectory $\tau$ is paired with the corresponding ground truth label $j^{\text{gt}}$ that indicates the position of the first erroneous step[2]. Given such labeled dataset, PRM is usually trained with the maximum likelihood objective:

$$\max_\theta \ \mathbb{E}_{(\tau, j^{\text{gt}})\sim\mathcal{D}} \log p_\theta(j = j^{\text{gt}}|\tau), \tag{1}$$

with the log-likelihood $\log p_\theta(j|\tau)$ defined as:

$$\log p_\theta(j|\tau) \coloneqq \mathbb{1}[j \leq T] \cdot \log r_\theta(c_j{=}0|\tau_{\leq j}) \ + \sum_{t<j} \log r_\theta(c_t{=}1|\tau_{\leq t}), \tag{2}$$

[1] In the literature, PRMs are sometimes defined as models that behave like value functions, estimating the probability that a partial trajectory will eventually yield a correct final answer rather than stepwise correctness. In this paper, we focus on step-level correctness as defined above.

[2] We follow such definition as the meaning of step's correctness may become ambiguous after the first erroneous step.

where the random variable $j \in \{1, \ldots, T, T+1\}$ represents the position of the first erroneous step in $\tau$, with $j = T+1$ indicating no error, and $\mathbb{1}[\cdot]$ corresponds to Iverson bracket.

### 3.2 Large Language Models as Scoring Functions

Pre-trained large language models (LLMs) can be repurposed to define scoring functions for downstream tasks by leveraging their next-token probabilities. In particular, given an LLM and a suitably constructed prompt, one can measure the plausibility of candidate solutions by examining and combining probabilities the model assigns to specific tokens.

For example, consider the task of verifying a biographical claim about Albert Einstein. Given the template $\mathcal{T}$ = "Albert Einstein won [award] in [year] for [contribution]" with candidates filled in, we can extract and sum probabilities at each position to define the score for a candidate triplet. More generally, extracting and blending probabilities at arbitrary positions within a templated sequence allows defining complex scoring functions $\mathcal{S}(a; \mathcal{T})$ that assess the plausibility of answers $a$. Intuitively, such scoring functions measure consistency with the knowledge acquired by the LLM during the pre-training stage.

Given such a score $\mathcal{S}(a; \mathcal{T})$, a policy $\pi_\theta$ can be trained to produce the most plausible answers via reinforcement learning:

$$\max_\theta \ \mathbb{E}_{\mathcal{T}\sim\mathcal{D}, a\sim\pi_\theta} \mathcal{S}(a; \mathcal{T}). \tag{3}$$

In the following section, we build on this principle to construct a score for training PRMs without access to ground-truth labels $j^{\mathrm{gt}}$.

## 4 Unsupervised Process Reward Models

Our goal is to train a PRM without relying on the curated labels $j^{\mathrm{gt}}$. The key idea is to define a scoring function derived from LLM next-token probabilities, which measures how plausible a candidate position of the first erroneous step is in a given trajectory. Subsequently, we train uPRM by optimizing this score, eliminating the need for any expert annotations.

### 4.1 Scoring First Erroneous Position with LLMs

Consider a trajectory $\tau = (x, y_1, \ldots, y_T)$ and a candidate position of the first erroneous step $j \in \{1, \ldots, T+1\}$. To define the scoring function, we interleave reasoning steps with correctness labels, marking steps $y_1, \ldots, y_{j-1}$ as correct and step $y_j$ as incorrect, resulting into a sequence:

$$\mathbf{s}(\tau, j) = [x,\ y_1,\ +,\ \ldots,\ y_{j-1},\ +,\ y_j,\ -], \tag{4}$$

where "+" and "−" denote correct and incorrect labels respectively. The special case $j = T+1$ (no error) corresponds to all steps marked as correct:

$$\mathbf{s}(\tau, T+1) = [x,\ y_1,\ +,\ \ldots,\ y_{j-1},\ +,\ y_T,\ +]. \tag{5}$$

We feed the constructed sequence to an LLM and extract the next-token probabilities LLM assigns to each label to define the scoring function $\mathcal{S}(j; \mathbf{s})$ as follows:

$$\mathcal{S}(j; \mathbf{s}) := \mathbb{1}[j \le T] \cdot \log p_j^- \ + \sum_{t<j} \log p_t^+, \tag{6}$$

where $p_t^+$ and $p_t^-$ denote the LLM's next-token probabilities of generating the label tokens "+" and "−" after $y_t$, respectively, renormalized over $\{+, -\}$.

### 4.2 Scoring Multiple Trajectories at Once

The score in Eq (6) can be viewed as an instance of the LLM-as-a-Judge paradigm (Zheng et al., 2023; Liu et al., 2023; Fu et al., 2024). Recent works have shown that LLMs produce more reliable judgments when evaluating multiple instances jointly rather than independently, whether through comparative ranking (Corbitt et al., 2025), batched evaluation (Korikov et al., 2025), or sequential

in-context learning (Gadetsky et al., 2025). Motivated by this, we extend our score to joint assessment of positions of first erroneous steps $j_1, \ldots, j_N,\ j_n \in \{1, \ldots, T_n + 1\}$ for a batch of $N$ trajectories $\tau_1, \ldots, \tau_N,\ \tau_n = (x, y_1, \ldots, y_{T_n})$.

To jointly score a batch of trajectories, we concatenate marked sequences $\mathbf{s}(\tau_n, j_n)$ together, obtaining:

$$\mathbf{s}_{1:N} = [\mathbf{s}(\tau_1, j_1),\ \ldots,\ \mathbf{s}(\tau_N, j_N)]. \tag{7}$$

Subsequently, the resulted sequence is fed to the LLM and the joint score is defined as:

$$\mathcal{S}(j_{1:N}; \mathbf{s}_{1:N}) = \frac{1}{N} \sum_{n=1}^{N} \Big( \mathbb{1}[j_n \leq T_n] \cdot \log p^{-}_{n,j_n} + \sum_{t<j_n} \log p^{+}_{n,t} \Big), \tag{8}$$

where $p^{+}_{n,t}$ and $p^{-}_{n,t}$ now denote the LLM's next-token probabilities of generating the corresponding label tokens for step $t$ in trajectory $\tau_n$, conditioned on all preceding tokens in $\mathbf{s}_{1:n}$, and renormalized over $\{+, -\}$ as before. We evaluate the reliability of the joint score through a controlled study in Appendix D.1, finding that it ranks more accurate first-error label configurations higher.

It is also worth noting that in this formulation, the score for a trajectory $\tau_n$ is computed given the previous trajectories $\tau_1, \ldots, \tau_{n-1}$ along with their candidate labels $j_1, \ldots, j_{n-1}$ as in-context examples. In practice, we observed a failure mode induced by this in-context learning effect. In particular, the joint score can become spuriously large for configurations in which all trajectories share the same label $j_n$, regardless of the actual error positions. We describe a simple correction that mitigates this effect in Appendix A.

### 4.3 Training PRM via Optimizing Joint Score

We parameterize PRM $r_\theta(c_t|\tau_{\leq t})$ by applying LoRA (Hu et al., 2022) to the same LLM used for computing the joint score. Noteworthy, this parametrization can be seen as an instantiation of self-training, in which a model trains by obtaining training signal from itself (Zelikman et al., 2022; Singh et al., 2024; Gadetsky et al., 2025). We follow recent best practices in model architectures to define PRMs (Zhang et al., 2025). In particular, given a trajectory $\tau = (x, y_1, \ldots, y_T)$, we construct a sequence by interleaving each reasoning step with a special token `[⋆]`:

$$[x,\ y_1,\ \texttt{[⋆]},\ y_2,\ \texttt{[⋆]},\ \ldots,\ y_T,\ \texttt{[⋆]}], \tag{9}$$

where the embedding of `[⋆]` is trainable. We process this sequence with the LLM and extract the last-layer hidden state $\mathbf{z}_t$ at each `[⋆]` token position following step $y_t$.

To obtain step-level correctness probabilities, we replace the language modeling head with a two-layer MLP with ReLU activation that projects each hidden state to two logits:

$$\mathbf{l}_t = \mathrm{MLP}(\mathbf{z}_t) \in \mathbb{R}^2, \tag{10}$$

which are converted to probabilities via softmax:

$$r_\theta(c_t = 1|\tau_{\leq t}) = \frac{\exp((\mathbf{l}_t)_1)}{\exp((\mathbf{l}_t)_0) + \exp((\mathbf{l}_t)_1)},\ r_\theta(c_t = 0|\tau_{\leq t}) = 1 - r_\theta(c_t = 1|\tau_{\leq t}). \tag{11}$$

The distribution over the position of the first erroneous step $p_\theta(j|\tau)$ is then defined as in Equation (2).

We train $p_\theta$ by optimizing the following entropy-regularized objective (Ziebart, 2010):

$$\max_\theta\ \mathbb{E}_{\{\tau_n\}_{n=1}^N \sim \mathcal{D}} \left[ \mathbb{E}_{j_n \sim p_\theta(\cdot|\tau_n)} \Big[ \mathcal{S}(j_{1:N}) \Big] + \frac{\gamma}{N} \sum_{n=1}^{N} \mathbb{H}(p_\theta(\cdot|\tau_n)) \right], \tag{12}$$

where $\mathbb{H}(\cdot)$ denotes Shannon entropy that prevents $p_\theta$ from premature convergence, and $\gamma$ corresponds to the regularization strength. We set $\gamma$ by monitoring the training curves and choosing the value that prevents collapse of $r_\theta$ throughout the training. We study the effect of $\gamma$ on the optimization in Appendix D.2.

**Efficient Optimization.** We develop a custom gradient estimator inspired by the actor-critic framework (Konda & Tsitsiklis, 1999) to enable efficient optimization of the objective (12). In particular,

on 8 H200 GPUs, uPRM training via our custom RL takes $\approx 5.5$ hours, compared to $\approx 4.25$ hours for supervised PRM trained via SFT on the same data and architecture, highlighting that the additional computational overhead is negligible relative to the expert labeling effort it removes. It is important to emphasize that joint scoring is used only during uPRM training. At test time, the trained uPRM processes trajectories independently, reflecting any existing PRM inference with no additional context length requirements. Thus, the overhead is a one-time training cost, not an inference cost. The details on the estimator are provided in Appendix B. Furthermore, rather than treating $N$ as the hyperparameter, we design a principled trajectory packing strategy that maximizes GPU memory utilization and ensures stable signal-to-noise ratio throughout training. We provide the details of this strategy in Appendix C.2.

# 5 EXPERIMENTS

**Method Instantiation.** We employ Qwen2.5-14B-Instruct (Yang et al., 2024a) to calculate the joint score in Eq (8) and instantiate the PRM $r_\theta$ in Eq (11). It is important to emphasize that Qwen2.5-14B-Instruct's post-training didn't involve training on any step-level correctness labels of reasoning chains, thus, keeping our setup fully unsupervised with respect to these labels. We ablate model size and model family in Appendix D.4 and Appendix D.5, finding that uPRM benefits from larger models and generalizes across model families.

We train uPRM on the PRM800K dataset (Lightman et al., 2024), using only the reasoning trajectories without any correctness labels. We use "+" and "−" as marker tokens in the joint scoring, and we find that uPRM is robust to their choice in Appendix D.6. The detailed description of the experimental setup and the implementation details are provided in Appendix C.

We evaluate uPRM along three dimensions. In Section 5.1, we directly assess its ability to detect step-level errors on the ProcessBench (Zheng et al., 2025). In Section 5.2, we use uPRM as a verifier, coupled with test-time scaling approaches, to measure its ability to guide inference. Finally, in Section 5.3, we use uPRM as a reward signal for reinforcement learning, demonstrating that it can effectively guide policy optimization.

## 5.1 PROCESSBENCH

We first evaluate the ability of uPRM to identify the position of the first erroneous step in reasoning trajectories as the most direct evaluation protocol. We employ ProcessBench (Zheng et al., 2025), a benchmark specifically designed to evaluate process reward models on step-level error detection. ProcessBench contains reasoning trajectories generated by various LLMs across four mathematical reasoning datasets of increasing difficulty: GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), OlympiadBench (He et al., 2024), and Omni-MATH (Gao et al., 2025). Each trajectory is annotated with the position of the first erroneous step, or marked as fully correct if no errors are present.

Following Zheng et al. (2025), we report three metrics: *(i)* accuracy on erroneous trajectories, measuring how often the model correctly identifies the first mistake in trajectories that contain errors; *(ii)* accuracy on correct trajectories, measuring how often the model correctly concludes that a trajectory is error-free; and *(iii)* F1 score computed as the harmonic mean of the two accuracies, which serves as the primary aggregated metric. We report F1 scores in Table 1 and provide the full breakdown in Table D4.

We compare uPRM with LLM-as-a-Judge (Eq 6) and LLM-as-a-Judge with a step-wise chain-of-thought (CoT) critic that reasons before predicting the first erroneous step (Zheng et al., 2025). It is important to note that such LLM-as-a-Judge (CoT) is much more inefficient compared to uPRM (Appendix D.3). To obtain a matched comparison, we additionally distill critic labels obtained on the PRM800K trajectories to a PRM under our exact setup, denoted as CoT-PRM. We also include fully-supervised PRM that is trained on ground truth step labels of PRM800 trajectories under our exact setup, denoted as sPRM, and current state-of-the-art Qwen2.5-Math-PRM-7B, to indicate the performance attainable with full supervision.

As shown in Table 1, uPRM consistently outperforms the LLM-as-a-Judge baseline across all four datasets by a large margin. The improvements are particularly pronounced on the more challenging

Table 1: Results on the ProcessBench dataset (F1 score).

| | ProcessBench | | | | |
|---|---|---|---|---|---|
| | GSM8K | MATH | OlympiadBench | Omni-MATH | Avg. |
| *Unsupervised methods* | | | | | |
| LLM-as-a-Judge | 49.8 | 42.8 | 29.4 | 26.6 | 37.2 |
| LLM-as-a-Judge (CoT) | 63.3 | 55.2 | 44.0 | 39.4 | 50.5 |
| CoT-PRM | 54.6 | 49.8 | 46.7 | 42.3 | 48.4 |
| uPRM (ours) | 58.3 | 52.6 | 42.7 | 39.8 | 48.4 |
| *Supervised methods* | | | | | |
| sPRM | 78.8 | 59.3 | 37.1 | 41.1 | 54.1 |
| Qwen2.5-Math-PRM-7B | 82.4 | 77.6 | 67.5 | 66.3 | 73.5 |

benchmarks: uPRM achieves a 13% absolute improvement on OlympiadBench and 13% on Omni-MATH. This suggests that training the PRM to optimize the joint score is especially beneficial when the underlying reasoning is more complex and the LLM's independent judgments are less reliable. These results confirm that our unsupervised training procedure successfully distills the evaluation capability of the LLM into a dedicated process reward model.

Although the LLM-as-a-Judge critic is stronger than uPRM at first-error localization, the performance gap narrows monotonically with benchmark difficulty, with uPRM performing slightly better on the most challenging Omni-MATH split. Furthermore, under a matched setup, the PRM distilled from CoT-critic labels performs similarly to uPRM, suggesting that uPRM provides a more compute-efficient path to comparable performance.

Most excitingly, uPRM remains competitive with fully supervised PRMs despite having no access to ground-truth step labels. It outperforms sPRM by 5.6% on OlympiadBench and lags behind it by only 1.3% points on Omni-MATH. This is particularly notable because these benchmarks contain more challenging reasoning trajectories than GSM8K and MATH.

### 5.2 Scaling Test-Time Compute with uPRM

We next evaluate the utility of uPRM in the test-time compute scaling paradigm (Lightman et al., 2024; Snell et al., 2025). We use Best-of-N and DVTS sampling strategies that require rewards from uPRM to guide and select the final response.

**Experimental Setup.** We evaluate a range of instruction-tuned LLMs across different parameter scales, including Qwen2.5-Instruct series (1.5B, 7B, 14B) (Yang et al., 2024a), Llama-3.2-1B-Instruct, and Llama-3.1-8B-Instruct (Grattafiori et al., 2024) to generate candidate responses. We set the generation temperature to 0.7 and use nucleus sampling with a cumulative probability threshold of 0.8. We define the test-time compute budget in terms of the number of independent generations, which we scale as powers of 2, up to 256 candidate answers per question. We assess the performance using accuracy on three standard mathematical benchmarks: MATH-500 (Hendrycks et al., 2021), MinervaMath (Lewkowycz et al., 2022), and Olympiad Bench (He et al., 2024). Finally, since PRM assigns step-level scores, we use the *last*-step score as the overall score for a candidate answer (Snell et al., 2025). We conduct an ablation study using an alternative aggregation method, where the *product* of step-wise scores is considered (Lightman et al., 2024; Zhang et al., 2025), as detailed in Appendix D.8. Our results indicate that the *last*-step score marginally outperforms the *product* score. For TTS experiments with uPRM, we run three independent seeds and report the mean performance.

**Results.** Figure 1 shows that uPRM assigns meaningful rewards to candidate responses, improving performance as the test-time compute budget increases. Notably, in Llama-3.2-1B-Instruct, the average accuracy across three benchmarks jumps from 14.6% (with 1 candidate response) to 31.7% (with 256 candidate responses), an absolute improvement of 17.1%. We also observe similar trends for uPRM initialized from Llama-3-70B in Appendix D.9. Next, we observe that the performance improvements depend on the sampling strategy, the policy LLM, and its size. For instance, DVTS with uPRM significantly outperforms other sampling strategies in smaller models, yielding 6.9% and 4.4% improvements over majority voting for Llama-3.2-1B-Instruct and Qwen2.5-1.5B-Instruct, respectively, and 2.8% and 1.5% improvements over Best-of-N, but Best-of-N outperforms DVTS for

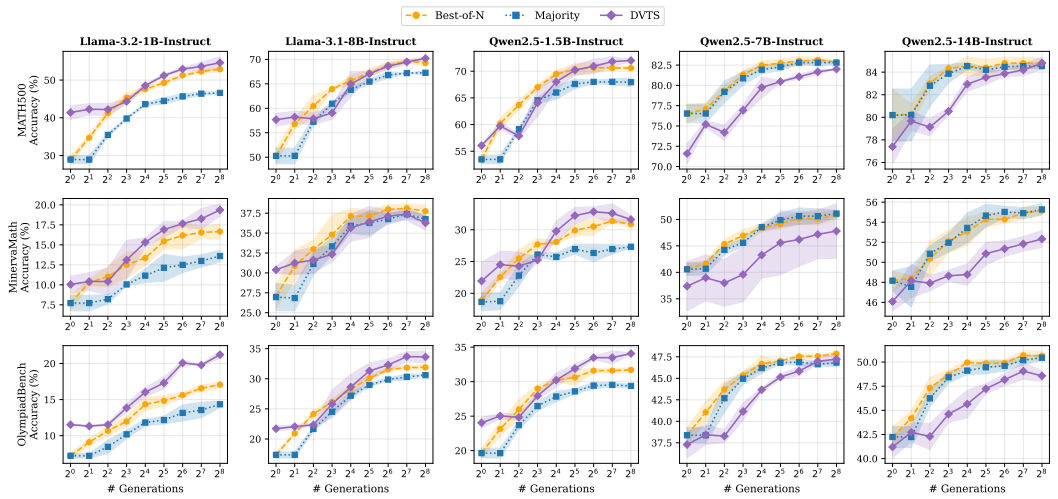



Figure 1: Accuracy of LLMs across different scales on MATH-500, MinervaMath, and Olympiad-Bench with different test-time scaling approaches on uPRM. Results are reported across three seeds.

larger policy LLMs. Similar findings about dependence on sampling strategies and non-generalization for PRMs have been studied before (Zhang et al., 2025; Liu et al., 2025a).

### 5.3 Reinforcement Learning with uPRM

Next, we explore whether uPRM obtained with our framework can be used as a reward source for reinforcement learning (RL). To this end, we adopt the PURE framework (Cheng et al., 2025) that incorporates dense rewards from PRMs into RL via min-form credit assignment. For a group of responses generated by a policy, PURE computes return values as an approximate minimum of per-step rewards and propagates them to per-token advantages, allowing PRM outputs to be naturally embedded into any policy gradient RL framework.

**Experimental Setup.** We adopt the experimental setup of Cheng et al. (2025) and perform RL fine-tuning of Qwen2.5 policy models (Yang et al., 2024a;b): Qwen2.5-7B, Qwen2.5-Math-7B, and Qwen2.5-Math-1.5B. Following Cheng et al. (2025), we use RLOO Ahmadian et al. (2024) as a training algorithm and a subset of hard (level 3–5) problems from the MATH dataset (Hendrycks et al., 2021) as training data. While the principled min-form credit assignment and other algorithmic adjustments introduced in PURE help delay reward hacking (RH), Cheng et al. (2025) argue that it is still inevitable when relying solely on PRM rewards. Therefore, as a preventive measure, the authors propose mixing per-step rewards with a standard outcome verifiable reward (VR) for a subset of the data to introduce an auxiliary ground-truth signal that will prevent overfitting on PRM. In line with these recommendations, we consider three training scenarios: *(i)* using VR only, *(ii)* using PRM rewards only, and *(iii)* using PRM + VR for $10\%$ of the data, as in Cheng et al. (2025). We keep hyperparameters the same for all training runs and vary only the reward source. Details can be found in Appendix C.3.

**Results.** We compare the performance of the RL-trained policies on MATH-500, MinervaMath, and Olympiad Bench in Table 2. Remarkably, uPRM is comparable to or better than VR and sPRM in learned-policy performance. For example, Qwen2.5-Math-1.5B trained with uPRM achieves a 4-point average accuracy gain across the three benchmarks compared with the same model trained using the ground-truth verifiable reward. In Appendix D.10, we provide an extended version of Table 2 that additionally compares against LLM-as-a-Judge and CoT-PRM as process reward models.

Interestingly, although Cheng et al. (2025) report that RH is inevitable for PRMs, we were able to successfully complete training of Qwen2.5-Math models using *just* rewards from uPRM and observed *no signs of RH*. In contrast, sPRM, which is the same PRM but trained via standard SFT, turned out to be highly susceptible to hacking: training with sPRM collapsed either almost immediately

Table 2: Accuracy on mathematical benchmarks after RL training with different reward sources. Entries report mean ± sample standard deviation across seeds. Cells shaded in green correspond to runs with our uPRM. Rows marked with † were evaluated at the last checkpoint before reward hacking, *i.e.*, prior to the end of training for at least one seed. For Qwen2.5-Math-7B, sPRM and sPRM+VR, and for Qwen2.5-7B, sPRM alone could not be evaluated due to rapid reward hacking.

| Model | Reward | MATH-500 | Minerva Math | OlympiadBench |
|---|---|---|---|---|
| Qwen2.5-7B | VR | $74.1_{\pm 0.8}$ | $34.2_{\pm 1.0}$ | $34.8_{\pm 1.0}$ |
| | sPRM + VR† | $\mathbf{75.4}_{\pm 0.0}$ | $29.4_{\pm 4.7}$ | $36.9_{\pm 0.4}$ |
| | uPRM† | $73.2_{\pm 0.4}$ | $35.0_{\pm 1.3}$ | $\mathbf{37.5}_{\pm 1.1}$ |
| | uPRM + VR† | $73.2_{\pm 1.4}$ | $\mathbf{35.8}_{\pm 0.6}$ | $35.7_{\pm 1.9}$ |
| Qwen2.5-Math-7B | VR | $80.1_{\pm 0.8}$ | $35.9_{\pm 0.4}$ | $41.8_{\pm 0.4}$ |
| | uPRM | $\mathbf{82.9}_{\pm 0.4}$ | $\mathbf{37.9}_{\pm 1.0}$ | $42.1_{\pm 1.3}$ |
| | uPRM + VR | $82.1_{\pm 1.3}$ | $36.3_{\pm 2.2}$ | $\mathbf{43.8}_{\pm 0.2}$ |
| Qwen2.5-Math-1.5B | VR | $70.0_{\pm 0.4}$ | $26.0_{\pm 0.2}$ | $33.5_{\pm 1.0}$ |
| | sPRM† | $\mathbf{74.7}_{\pm 0.3}$ | $27.8_{\pm 0.6}$ | $35.0_{\pm 1.0}$ |
| | sPRM + VR† | $74.4_{\pm 1.2}$ | $28.7_{\pm 1.3}$ | $36.3_{\pm 0.6}$ |
| | uPRM | $73.5_{\pm 1.2}$ | $\mathbf{31.8}_{\pm 0.8}$ | $\mathbf{36.6}_{\pm 0.6}$ |
| | uPRM + VR | $74.3_{\pm 0.5}$ | $31.5_{\pm 0.8}$ | $35.8_{\pm 0.5}$ |

($< 50$ iterations) for Qwen2.5-Math-7B or after several hundreds of training iterations for the smaller 1.5B model. For the Qwen2.5-7B base model, where both PRMs succumbed to RH before training completed, we observe qualitatively different hacking behaviors in the learned policy under uPRM versus sPRM. We provide a detailed analysis of this phenomenon in Appendix D.10.

## 6 Conclusion and Limitations

In this work, we propose a *fully unsupervised approach* for training PRMs that requires neither step-level annotations nor ground-truth verification of final answers. Our experiments demonstrate that our unsupervised PRM is competitive to supervised PRMs trained with expert annotations, thus, the marginal cost of obtaining step-wise guidance for new domains, model families, or inference/training pipelines can be significantly reduced. Notable experimental evidence is that strong downstream utility does not require perfect localization of erroneous steps. Indeed, while our uPRM may lag behind state-of-the-art supervised PRMs on error localization benchmarks such as ProcessBench, it remains competitive in settings where PRMs actually provide value. In particular, these include serving as verifiers in test-time scaling and as a reward source in reinforcement learning. In general, our results reinforce the view that the direct accuracy of a reward model is an incomplete proxy for downstream utility, consistent with recent work (Razin et al., 2025) that finds that the most accurate reward models are not necessarily the most effective teachers.

**Limitations.** As we justify in Section 5.1, joint scoring is the crucial component for training a strong PRM. Since it requires an LLM with sufficient context length to process concatenated trajectory batches and sufficient capability to produce meaningful correctness judgments, it limits the choice of base models. Both limitations can be mitigated by decoupling the scoring LLM from the PRM backbone, allowing usage of more capable model to provide training signal while a smaller model serves as the final PRM. Moreover, as context windows and capabilities of open-source LLMs continue to grow, these constraints will naturally relax.

Beyond introducing the first fully unsupervised PRM training method, our paper identifies a practically important robustness phenomenon (Section 5.3) that had not been highlighted before. Although we make an effort to unravel the source of this robustness (Appendix D.10.1), fully characterizing its precise origin is an important next step, and we see this as a promising direction opened by our work for the broader community.

## References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, et al. Back to Basics: Revisiting REINFORCE-Style Optimization for Learning from Human Feedback in LLMs.

In *Association for Computational Linguistics*, 2024.

Lisa Alazraki, Maximilian Mozes, Jon Ander Campos, Tan Yi-Chern, Marek Rei, et al. No Need for Explanations: LLMs Can Implicitly Learn from Mistakes In-context. *Conference on Empirical Methods in Natural Language Processing*, 2025.

Edward Beeching, Lewis Tunstall, and Sasha Rush. Scaling Test-Time Compute with Open Models, 2024. URL https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, et al. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. AlphaMath Almost Zero: Process Supervision without Process. *Neural Information Processing Systems*, 2024.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, et al. Towards Reasoning Era: A Survey of Long Chain-of-Thought for Reasoning Large Language Models. *arXiv preprint arXiv:2503.09567*, 2025.

Jie Cheng, Gang Xiong, Ruixi Qiao, Lijun Li, Chao Guo, et al. Stop Summation: Min-Form Credit Assignment Is All Process Reward Model Needs for Reasoning. In *Neural Information Processing Systems*, 2025.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, et al. SFT Memorizes, RL Generalizes: A Comparative Study of Foundation Model Post-training. In *International Conference on Machine Learning*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, et al. Training Verifiers to Solve Math Word Problems. *OpenAI Technical Report*, 2021.

Kyle Corbitt, Saumya Gandhi, Angky William, Andie Jones, Brad Hilton, et al. RULER: Relative Universal LLM-Elicited Rewards. *OpenPipe Blog*, 2025.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, et al. Process Reinforcement Through Implicit Rewards. *arXiv preprint arXiv:2502.01456*, 2025.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. FlexAttention: A Programming Model for Generating Fused Attention Variants. In *Conference on Machine Learning and Systems*, 2025.

Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, et al. Kimi k1.5: Scaling Reinforcement Learning with LLMs. *arXiv preprint arXiv:2501.12599*, 2025.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. Length-Controlled AlpacaEval: A Simple Way to Debias Automatic Evaluators. In *Conference on Language Modeling*, 2024.

Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. GPTScore: Evaluate as You Desire. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2024.

Artyom Gadetsky, Andrei Atanov, Yulun Jiang, Zhitong Gao, Ghazal Hosseini Mighan, et al. Large (Vision) Language Models are Unsupervised In-Context Learners. In *International Conference on Learning Representations*, 2025.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, et al. Omni-MATH: A Universal Olympiad Level Mathematic Benchmark for Large Language Models. In *International Conference on Learning Representations*, 2025.

Leo Gao, John Schulman, and Jacob Hilton. Scaling Laws for Reward Model Overoptimization. In *International Conference on Machine Learning*, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, et al. DeepSeek-R1 Incentivizes Reasoning in LLMs through Reinforcement Learning. *Nature*, 2025.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, et al. OlympiadBench: A Challenging Benchmark for Pomoting AGI with Olympiad-level Bilingual Multimodal Scientific Problems. In *Association for Computational Linguistics*, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, et al. Measuring Mathematical Problem Solving With the MATH Dataset. In *Neural Information Processing Systems*, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, et al. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, et al. VinePPO: Refining Credit Assignment in RL Training of LLMs. In *International Conference on Machine Learning*, 2025.

Vijay Konda and John Tsitsiklis. Actor-Critic Algorithms. In *Neural Information Processing Systems*, 1999.

Anton Korikov, Pan Du, Scott Sanner, and Navid Rekabsaz. Batched Self-Consistency Improves LLM Relevance Assessment and Ranking. In *Conference on Empirical Methods in Natural Language Processing*, 2025.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, et al. Solving Quantitative Reasoning Problems with Language Models. *Neural Information Processing Systems*, 2022.

Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, et al. From System 1 to System 2: A Survey of Reasoning Large Language Models. *arXiv preprint arXiv:2502.17419*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, et al. Let's Verify Step by Step. In *International Conference on Learning Representations*, 2024.

Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, et al. Can 1B LLM Surpass 405b LLM? Rethinking Compute-Optimal Test-Time Scaling. *arXiv preprint arXiv:2502.06703*, 2025a.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, et al. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, et al. Understanding R1-Zero-Like Training: A Critical Perspective. In *Conference on Language Modeling*, 2025b.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019.

Jianqiao Lu, Zhiyang Dou, Hongru Wang, Zeyu Cao, Jianbo Dai, et al. AutoPSV: Automated Process-Supervised Verifier. *Neural Information Processing Systems*, 2024.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, et al. Improve Mathematical Reasoning in Language Models by Automated Process Supervision. *arXiv preprint arXiv:2406.06592*, 2024.

Ruilin Luo, Zhuofan Zheng, Yifan Wang, Xinzhe Ni, Zicheng Lin, et al. URSA: Understanding and Verifying Chain-of-thought Reasoning in Multimodal Mathematics. *arXiv preprint arXiv:2501.04686*, 2025.

Noam Razin, Zixuan Wang, Hubert Strauss, Stanley Wei, Jason D. Lee, et al. What Makes a Reward Model a Good Teacher? An Optimization Perspective. In *Neural Information Processing Systems*, 2025.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, et al. Rewarding Progress: Scaling Automated Process Verifiers for LLM Reasoning. In *International Conference on Learning Representations*, 2025.

Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. RL's Razor: Why Online Reinforcement Learning Forgets Less. *arXiv preprint arXiv:2509.04259*, 2025.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, et al. HybridFlow: A Flexible and Efficient RLHF Framework. In *European Conference on Computer Systems*, 2025.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, et al. Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models. *Transactions on Machine Learning Research*, 2024.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning. In *International Conference on Learning Representations*, 2025.

Lin Sun, Chuang Liu, Xiaofeng Ma, Tao Yang, Weijia Lu, and Ning Wu. FreePRM: Training Process Reward Models Without Ground Truth Process Labels. *arXiv preprint arXiv:2506.03570*, 2025.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, et al. Solving Math Word Problems with Process- and Outcome-based Feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, et al. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. In *Association for Computational Linguistics*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Neural Information Processing Systems*, 2022.

Ronald J. Williams. Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 1992.

Bin Xie, Bingbing Xu, Yige Yuan, Shengmao Zhu, and Huawei Shen. From Outcomes to Processes: Guiding PRM Learning from ORM for Inference-Time Alignment. In *Association for Computational Linguistics*, 2025.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*, 2024a.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, et al. Qwen2.5-Math Technical Report: Toward Mathematical Expert Model via Self-Improvement. *arXiv preprint arXiv:2409.12122*, 2024b.

Jiarui Yao, Ruida Wang, and Tong Zhang. PRL: Process Reward Learning Improves LLMs' Reasoning Ability and Broadens the Reasoning Boundary. *arXiv preprint arXiv:2601.10201*, 2026.

Zhangyue Yin, Qiushi Sun, Zhiyuan Zeng, Qinyuan Cheng, Xipeng Qiu, et al. Dynamic and Generalizable Process Reward Modeling. In *Association for Computational Linguistics*, 2025.

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, et al. Free Process Rewards without Process Labels. In *International Conference on Machine Learning*, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Self-Taught Reasoner Bootstrapping Reasoning With Reasoning. *Neural Information Processing Systems*, 2022.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, et al. The Lessons of Developing Process Reward Models in Mathematical Reasoning. In *Association for Computational Linguistics*, 2025.

Jian Zhao, Runze Liu, Kaiyan Zhang, Zhimu Zhou, Junqi Gao, et al. GenPRM: Scaling Test-Time Compute of Process Reward Models via Generative Reasoning. *arXiv preprint arXiv:2504.00891*, 2025.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, et al. ProcessBench: Identifying Process Errors in Mathematical Reasoning. In *Association for Computational Linguistics*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wuand, et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Neural Information Processing Systems*, 2023.

Brian Ziebart. Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy. *Carnegie Mellon University*, 2010.

## A Score Correction to Mitigate Degenerate Solutions

In our preliminary experiments, we observed that, although, the score usually assigns higher values to configurations of $j_1, \ldots, j_N$ that are close to ground truth as desired, it also encourages degenerate solutions due to in-context learning pathologies. We observed at least 2 degenerate solutions: setting each $j_n = 1$ or setting each $j_n = T_n + 1$, corresponding to the first step labeled as erroneous in each trajectory and the absence of erroneous steps in each trajectory, respectively. We add the correction term to our joint score, excluding the aforementioned configurations. In particular, let $W_n = 1 + \log(\sqrt{T_n + 1})$. Intuitively, $W_n$ measures the amount of surprise when observing the realized value of $j_n$. Let $S_{\text{first}}(j_{1:N}) = \sum_{n=1}^{N} W_n \cdot \mathbb{1}[j_n = 1]$, $S_{\text{last}}(j_{1:N}) = \sum_{n=1}^{N} W_n \cdot \mathbb{1}[j_n = T_n + 1]$, and $S_{\text{corner}}(j_{1:N}) = S_{\text{first}}(j_{1:N}) + S_{\text{last}}(j_{1:N})$. Let $S_{\max} = \sum_{n=1}^{N} W_n$ and $B = (1 - \rho)S_{\max}$ for $\rho \in (0, 1)$, where $B$, intuitively, signifies the allowed surprise budget. Consequently, $\rho$ defines the amount of non-corner values of $j_n$'s that are allowed to happen in the group $j_1, \ldots, j_N$. As a result, our correction term is defined as:

$$S_{\text{correction}}(j_{1:N}) = -\max(0, S_{\text{corner}}(j_{1:N}) - B), \tag{13}$$

where we found $\rho = 0.25$ is the good default choice. It corresponds to the budget $B = 0.75 \cdot S_{\max}$, thus, the regularization is inactive unless more than 75% of the batch predictions collapse to the corner categories. This makes it a weak safeguard against severe collapse rather than a strong bias against general corner predictions.

We also conducted additional training runs with different values of $\rho$ to directly assess the robustness of our correction term. The results in Figure A1 show no significant differences in either $p_\theta(\cdot|\tau)$ entropy or model performance.

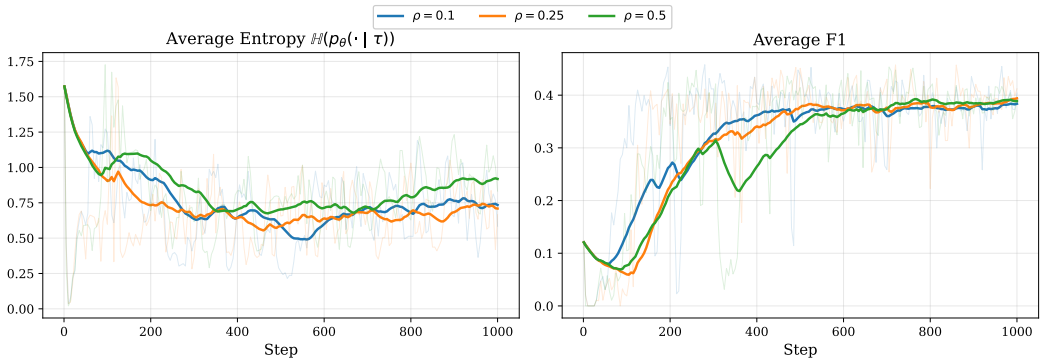



Figure A1: Ablation of the non-corner budget in the correction term.

Consequently, we can redefine our joint score as follows:

$$\mathcal{S}(j_{1:N}) = \mathcal{S}(j_{1:N}; \mathbf{s}_{1:N}) \; + \frac{1}{N} S_{\text{correction}}(j_{1:N}),$$

where $\mathcal{S}(j_{1:N}; \mathbf{s}_{1:N})$ is defined in Equation (8).

# B Optimization Details

## B.1 Gradient Estimator

Recall our entropy-regularized objective from Equation (12):

$$\max_{\theta} \; \mathbb{E}_{\{\tau_n\}_{n=1}^N \sim \mathcal{D}} \left[ \mathbb{E}_{j_n \sim p_\theta(\cdot|\tau_n)} \mathcal{S}(j_{1:N}) \right] + \frac{\gamma}{N} \sum_{n=1}^{N} \mathbb{H}(p_\theta(\cdot|\tau_n)) \Bigg]. \tag{14}$$

Given that the gradients for the entropy term $\mathbb{H}(p_\theta(\cdot|\tau_n))$ can be easily computed using standard automatic differentiation engines, it is only required to derive the unbiased gradient estimator for the first term:

$$\mathcal{J}(\theta) = \mathbb{E}_{\{\tau_n\}_{n=1}^N \sim \mathcal{D}} \mathbb{E}_{j_n \sim p_\theta(\cdot|\tau_n)} \mathcal{S}(j_{1:N}) = \mathbb{E}_{\{\tau_n\}_{n=1}^N \sim \mathcal{D}} \mathbb{E}_{j_n \sim p_\theta(\cdot|\tau_n)} \sum_{n=1}^{N} \mathcal{S}(j_n|j_{<n}), \tag{15}$$

where, for notational brevity, we write $\mathcal{S}(j_{1:N}) = \sum_{n=1}^{N} \mathcal{S}(j_n|j_{<n})$ since our joint score admits autoregressive factorization. Let's introduce the following variables:

$$G_m = \sum_{n=m}^{N} S(j_n|j_{<n}), \; G_{N+1} = 0, \tag{16}$$

$$b_m^{\text{imm}}(j_{<m}) = \mathbb{E}_{j_m \sim p_\theta(\cdot|\tau_m)} S(j_m|j_{<m}) = \sum_{j=1}^{T_m+1} \mathcal{S}(j_m = j|j_{<m}) p_\theta(j_m = j|\tau_m), \tag{17}$$

where one can note that it is possible to efficiently compute $b_m^{\text{imm}}$ for all $m$ in a single forward pass of an LLM using custom FlexAttention (Dong et al., 2025) masks. The gradient estimator, inspired by the actor-critic framework (Konda & Tsitsiklis, 1999), is defined as follows:

$$\nabla_\theta \mathcal{J}(\theta) = \sum_{m=1}^{N} \left[ \left[ S(j_m|j_{<m}) - b_m^{\text{imm}}(j_{<m}) \right] + (G_{m+1} - \mathcal{V}_\phi(j_{<m})) \right] \nabla_\theta \log p_\theta(j_m|\tau_m), \tag{18}$$

where $\mathcal{V}_\phi(j_{<m})$ is a trainable critic neural network parametrized by $\phi$, and is allowed to depend on $j_{<m}$ without introducing bias into the estimator. The gradient estimator has the form of REINFORCE (Williams, 1992) with the baseline for variance reduction that directly computes the optimal immediate part $b_m^{\text{imm}}(j_{<m})$ and employs the neural network critic $\mathcal{V}_\phi(j_{<m})$ to estimate future returns $G_{m+1}$. To reduce the variance of the estimator even further, the critic is trained to approximate the returns $G_{m+1}$:

$$\mathcal{L}_{\text{critic}}(\phi) = \frac{1}{N-1} \sum_{m=1}^{N-1} (G_{m+1} - \mathcal{V}_\phi(j_{<m}))^2. \tag{19}$$

In the following section, we describe the architecture of the critic network.

## B.2 Critic Architecture

The critic $\mathcal{V}_\phi(j_{<m})$ must estimate future returns $G_{m+1} = \sum_{n=m+1}^{N} \mathcal{S}(j_n|j_{<n})$ given the history of sampled positions $j_{<m}$. We design the critic architecture with two considerations in mind: ***(i)*** avoiding additional LLM forward passes by reusing hidden states already computed during the joint score and PRM calculation, and ***(ii)*** leveraging privileged information about future trajectories $\tau_{m+1}, \ldots, \tau_N$ to facilitate estimation of future returns.

Specifically, we extract two types of hidden representations:

- From the joint score computation (8), we collect the last-layer hidden state at the final token of each marked sequence $\mathbf{s}(\tau_n, j_n)$, denoted $\mathbf{h}_n \in \mathbb{R}^d$. These representations encode the history of trajectories and their sampled positions, *i.e.*, $\mathbf{h}_n = \mathbf{h}_n(j_{\leq n})$. We additionally define $\mathbf{h}_0$ as the hidden state at the end of the system prompt, before any trajectory is processed.

- From the PRM forward pass, we extract the hidden state at the final special token for each trajectory $\tau_n$, denoted $\mathbf{g}_n \in \mathbb{R}^d$. These representations serve as trajectory embeddings and constitute privileged information available only during training.

The critic combines these representations using cross-attention. Let $H = [\mathbf{h}_0, \ldots, \mathbf{h}_{N-1}] \in \mathbb{R}^{N\times d}$ and $G = [\mathbf{g}_1, \ldots, \mathbf{g}_N] \in \mathbb{R}^{N\times d}$. To compute $\mathcal{V}_\phi(j_{<m})$, we first obtain a contextualized representation of future trajectories:

$$Q_m = W_q \mathbf{h}_{m-1}, \tag{20}$$

$$K = GW_k, \quad V = GW_v, \tag{21}$$

$$\alpha_m = \mathrm{softmax}\left(\frac{KQ_m}{\sqrt{d}}\right), \tag{22}$$

$$C_m = \sum_{n=1}^{N} (\alpha_m)_n V_n, \tag{23}$$

where $W_q, W_k, W_v \in \mathbb{R}^{d\times d}$ are learnable parameters. The contextualized representation $C_m$ aggregates information about all trajectories, weighted by their relevance to the current history $\mathbf{h}_{m-1}$. Finally, the critic value is computed as:

$$\mathcal{V}_\phi(j_{<m}) = \mathrm{MLP}([\mathbf{h}_{m-1}; C_m]), \tag{24}$$

where $[\cdot;\cdot]$ denotes concatenation and $\mathrm{MLP} : \mathbb{R}^{2d} \to \mathbb{R}$ is a two-layer network with GELU activation between the layers. Intuitively, $\mathbf{h}_{m-1}$ provide the critic with representations conditioned on $j_{<m}$, while contextualized representations $C_m$ equip the critic with the privileged information about future trajectories, enabling accurate prediction of $G_{m+1}$. In practice, we employ multi-head cross-attention with $8$ heads and project the hidden states from dimension $d$ to a hidden dimension of $1024$. The attention output is passed through an output projection $W_o$, followed by dropout (with probability $0.1$) and layer normalization. The final value is computed by concatenating the normalized history $\mathbf{h}_{m-1}$ with the attention context $C_m$ and passing through a two-layer MLP with GELU activation. We apply layer normalization to $H$ and $G$ before the cross-attention, and do not backpropagate gradients through the hidden states. The learnable parameters are $\phi = \{W_q, W_k, W_v, W_o, \mathrm{MLP}\}$.

# C Experimental and Implementation Details

## C.1 Prompt Templates

We use Qwen2.5-14B-Instruct as our LLM, leveraging its chat format with system and user/assistant turns. Below, we describe the prompt templates used for the joint score and PRM computation.

**System prompt.** We use the following system prompt during both the joint score and PRM calculations:

System Prompt

```
You are a strict mathematical reasoning judge.

Your task is to evaluate one individual reasoning step of a math
problem at a time.

- If the step is mathematically correct, respond with `+`.
- If the step is mathematically incorrect or logically flawed,
respond with `-`.
- Do not provide any explanation, comment, or feedback - only
respond with `+` or `-`, and nothing else.
- Each input is either a single reasoning step or a new problem
followed by its first reasoning step.  In both cases, evaluate only
the validity of the reasoning step.
- For each new problem, once you determine that a step is incorrect,
you must consider all subsequent steps for that problem to also be
incorrect, and respond with `-` for them as well.

Your response must only be one of these two symbols:  `+` or `-`.
```

**Conversation structure.** We format each trajectory as a multi-turn conversation where the user provides reasoning steps and the assistant responds with markers. Figure C1 illustrates the template structure.

<|im_start|>system
*[System prompt as above]* <|im_end|>
<|im_start|>user
*[Problem $x$] [Step $y_1$]* <|im_end|>
<|im_start|>assistant
*[Marker]* <|im_end|>
<|im_start|>user
*[Step $y_2$]* <|im_end|>
<|im_start|>assistant
*[Marker]* <|im_end|>
⋮

Figure C1: **Prompt template structure.** For the joint score, markers are +/− tokens, and multiple trajectories are concatenated sequentially, with each new problem introduced in the user turn. The conversation for trajectory $\tau_n$ terminates at step $j_n$ with marker − (or continues through all steps with + if $j_n = T_n + 1$). For the PRM, markers are the special token `[⋆]` and all steps are included. Furthermore, each trajectory is processed independently.

### C.2 Hyperparameters for Training uPRM

We apply LoRA (Hu et al., 2022) to an LLM, attaching low-rank adapters to all linear layers in the transformer. We set LoRA rank to 64, scaling factor $\alpha = 32$, and disable both bias terms and dropout. The trainable parameters consist of the LoRA adapters, the embedding of the special token `[⋆]`, and the two-layer MLP that projects hidden states to step-level probabilities.

We train with the AdamW optimizer (Loshchilov & Hutter, 2019) using a constant learning rate of $10^{-5}$ for 1000 gradient updates across 8 H200 GPUs with 8 gradient accumulation steps. Each device at each gradient accumulation step processes a single batch of trajectories, resulting in effective batch size of 64.

Rather than fixing the number of trajectories $N$ per batch, we pack trajectories in random order on each GPU such that the total number of reasoning steps equals 80, resulting in approximately $N = 13$

trajectories on average. This number corresponds to the maximal that fits into GPU memory without causing out-of-memory errors in our setting. If the last trajectory in a batch exceeds the remaining budget, we truncate it. This remains valid since our objective is to identify the position of the first mistake, and truncation only removes later steps. Importantly, fixing the total number of steps rather than fixing $N$ is a deliberate design choice. Indeed, since our gradient estimator (Appendix B.1) operates at the level of individual step predictions, fixing $N$ would result in a variable number of steps across batches due to different trajectory lengths, leading to fluctuating signal-to-noise ratio (SNR) in the gradients. By fixing the total step count instead, each batch contributes a consistent number of step-level predictions to the objective function, ensuring stable SNR throughout training.

### C.3 Reinforcement Learning with PRM

To perform RL with dense PRM rewards, we integrated our PRMs into an open-source PURE[3] (Cheng et al., 2025) implementation based on VeRL (Sheng et al., 2025). We used the same hyperparameters as suggested in the original work, detailed in Table C1. For each reward–policy combination, we conduct three independent runs with different random seeds.

For the PRM + VR setting, we did not vary the coefficients before the PRM term and the VR term and set both to 1, which results in a plain sum of the two terms. We also disabled the curriculum learning option available in the latest PURE implementation to avoid any confounding factors and stay as close to the original method as possible.

During evaluation, we apply greedy decoding (sampling temperature 0). In Table 2, the results are aggregated over the last available non-degenerate models *i.e.*, either at the end of training or at the last saved checkpoint before RH. Qwen2.5-7B with sPRM + VR is reported over 2 seeds because one trial failed before reaching the first checkpoint.

Table C1: Training hyperparameters and optimization settings for RL experiments. For Qwen2.5-Math models, we set a smaller generation length of 4096 due to the limited context.

| **Hyperparameter** | **Value** |
|---|---|
| Epochs | 4 (532 iterations) |
| Learning rate | $10^{-6}$ (constant) |
| Prompt batch size | 64 |
| Group size (responses per prompt) | 8 |
| Mini-batch size | 512 |
| Maximum generation length (tokens) | 8192 |
| Sampling temperature | 1.0 |
| KL coefficient | $10^{-3}$ |
| PURE transform temperature | 0.1 |
| Save interval (iterations) | 50 |

## D Additional Results

### D.1 Reliability of Joint Scoring

Here, we study whether the joint scoring objective emits reliable signal where more accurate labelings receive higher scores. To test this property, we consider 100 random batches of $N$ reasoning trajectories from the PRM800K dataset ($N$ varies between 11 and 15, mirroring batch composition during training) and pack them together for the joint scoring. For a given batch, for each $k = 0, \ldots, N$ we select $k$ trajectories uniformly at random and resample each of their ground truth first-error position to the incorrect alternative (i.e. from $\{0, \ldots, T_n\} \setminus \{j_n^*\}$ and compute the joint score under the Qwen2.5-14B-Instruct judge (Eq (8)). In simple terms, we would like to see if the joint scoring assigns higher score to configurations with more correct $j^*$'s for trajectories within a batch. We repeat this across 100 random batches and find Pearson correlation of $-0.995 \pm 0.004$ between number of corrupted trajectories $k$ and the joint score averaged over these 100 batches, the std is a bootstrap

[3] https://github.com/CJReinforce/PURE

over batches. Furthermore, in paired comparison, the ground truth labeling outscores a randomly corrupted one in 89% of cases (chance level — 50%). These results confirm that the joint score provides a faithful proxy to identify correctness of steps.

### D.2 Ablation of Entropy Regularization Strength

We study the effect of the entropy regularization strength $\gamma$ on the optimization dynamics. Figure D1 shows the average entropy $\mathbb{H}(p_\theta(\cdot|\tau))$ and average joint score $\mathcal{S}(j_{1:N})$ throughout training for three values of $\gamma \in \{3^0, 3^1, 3^2\}$.

When $\gamma$ is too small ($\gamma = 1$), the entropy collapses rapidly, dropping to near zero by step 400. This premature collapse indicates that $p_\theta$ converges to near-deterministic predictions early in training, losing the ability to explore alternative positions of the first erroneous step. While this leads to the highest joint scores, the resulting PRM may overfit to spurious patterns in the scoring function rather than learning robust error detection. Conversely, when $\gamma$ is too large ($\gamma = 9$), $p_\theta$ stays nearly uniform over candidate positions, preventing it from exploiting the signal in the joint score. The intermediate value ($\gamma = 3$) provides a favorable trade-off between exploration and exploitation. The entropy decreases gradually, allowing the model to concentrate probability mass on plausible error positions while maintaining sufficient exploration to avoid premature convergence. Based on this analysis, we use $\gamma = 3$ for all experiments reported in the main paper.

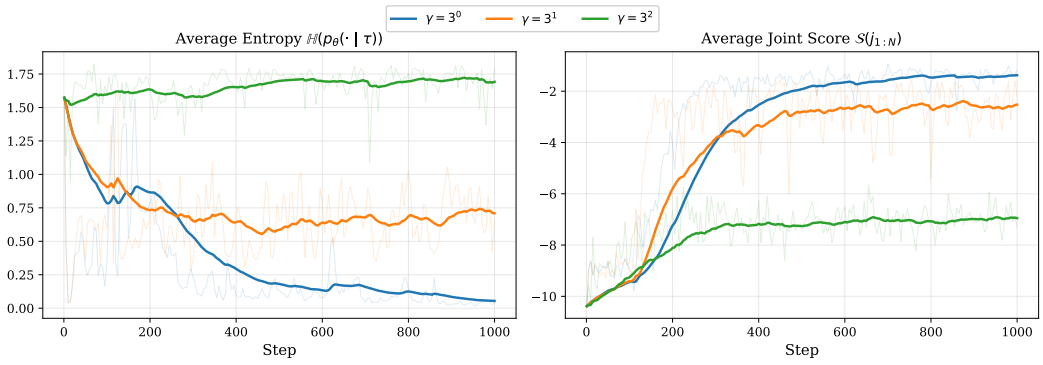



Figure D1: Ablation of the entropy regularization strength $\gamma$. Low values cause premature entropy collapse and overoptimization of the joint score, while $\gamma = 3$ keeps unsupervised PRM $p_\theta(\cdot|\tau)$ non-collapsed throughout the training, balancing exploration and exploitation.

### D.3 Cost of LLM-as-a-Judge Chain-of-Thought Critic Compared to uPRM

We would like to highlight that the inference cost of LLM-as-a-Judge with Chain-of-Thought (CoT) reasoning is much higher compared to any well-established PRM since it requires generating CoT compared to a single forward pass to label the reasoning trajectory. In particular, a PRM scores an entire trajectory of $L$ input tokens in a single forward pass, resulting in $\approx 2 \times N \times L$ FLOPs for an $N$-parameter model under standard transformer FLOP accounting. A CoT critic must first read the same input and then autoregressively generate a critique of $G$ tokens before emitting a verdict, resulting in $\approx 2 \times N \times (L + G)$ FLOPs and, crucially, $G$ strictly sequential decode steps, since each generated token depends on the previous one. To instantiate these estimates, we computed token statistics on PRM800K training data. The CoT critic generates on average $G \approx 306$ critique tokens per trajectory for an average input of $L \approx 444$ tokens. Per judgment, this is $\approx 1.7\times$ the FLOPs of a PRM forward pass, but the latency gap is far larger. Each decode step produces a single token yet requires a full read of the model weights from memory, so one critique performs 307 weight reads (306 decode steps plus prefill) versus a single read for a PRM pass. This yields 9 TB versus $\approx 30$ GB of weight traffic per judgment for a 14B model in bf16. Since decoding is memory-bandwidth-bound at all but very large batch sizes, this yields an idealized lower bound of $\approx 1.9$ s per judgment at H200 peak bandwidth (4.8 TB/s), versus 10-30 ms for one PRM forward pass, resulting in a $> 60\times$ latency gap.

### D.4 uPRM Benefits Scaling Model Size

To evaluate the effect of model scale, we train uPRM using a larger 32B model based on Qwen2.5-32B-Instruct (Yang et al., 2024a) while following the training setup used in our main experiments. As shown in Table D1, scaling from 14B to 32B consistently improves performance across all ProcessBench splits, yielding gains up to 8.6%. These results demonstrate that uPRM benefits substantially from scaling model size.

Table D1: ProcessBench results for different uPRM model sizes (F1 score).

| | GSM8K | MATH | OlympiadBench | Omni-MATH |
|---|---|---|---|---|
| uPRM-14B | 58.3 | 52.6 | 42.7 | 39.8 |
| uPRM-32B | **66.9** | **58.4** | **49.2** | **48.0** |

### D.5 uPRM Generalizes Across Model Families

To examine whether uPRM training transfers across model families, we train a uPRM based on Llama3.3-70B-Instruct (Grattafiori et al., 2024). The results in Table D2 show that the resulting uPRM outperforms the naive LLM-as-a-Judge baseline across all ProcessBench splits by up to 4.0%. These results suggest that the effectiveness of uPRM training is not specific to the Qwen family used in the main paper and extends to other large language model backbones.

Table D2: ProcessBench results using Llama3.3-70B-Instruct as the backbone (F1 score).

| Method | GSM8K | MATH | OlympiadBench | Omni-MATH |
|---|---|---|---|---|
| uPRM | **64.4** | **52.2** | **49.1** | **42.4** |
| LLM-as-a-Judge (naive) | 63.4 | 48.2 | 45.6 | 41.2 |

### D.6 Ablation of Marker Tokens

Joint scoring depends on the choice of marker tokens for correct and incorrect labels. We conduct ablation of marker tokens and run uPRM training with "`correct`" and "`incorrect`" instead of "+" and "−", adopting the system prompt from Appendix C.1 accordingly. The evaluation of both uPRMs on the ProcessBench benchmark in Table D3 confirms that uPRM is robust to the choice of marker tokens.

Table D3: Ablation of marker tokens in the joint scoring for uPRM training.

| | **GSM8K** | **MATH** | **OlympiadBench** | **Omni-MATH** |
|---|---|---|---|---|
| +/− | 58.3 | 52.6 | 42.7 | 39.8 |
| `correct`/`incorrect` | 57.8 | 48.3 | 40.9 | 39.0 |

### D.7 Complete Results on ProcessBench

Table D4 presents the complete breakdown of results on the ProcessBench benchmark (Zheng et al., 2025), including accuracy on erroneous trajectories (Err.), accuracy on correct trajectories (Corr.), and their harmonic mean (F1). We additionally report results on the the PRM800K dataset used for training, which serves as a sanity check since our unsupervised PRM is expected to perform better than LLM-as-a-Judge given it is trained on these trajectories.

The results indicate that our unsupervised PRM consistently outperforms the LLM-as-a-Judge baseline across all datasets and metrics, achieving absolute F1 improvements ranging from +8% on the GSM8K dataset to +14% on the OlympiadBench dataset. This confirms that optimizing the joint score successfully distills the LLM's evaluation capability into a more effective process reward model.

**Accuracy on Erroneous Trajectories.** Detecting errors in flawed reasoning is generally more challenging than recognizing correct solutions. The LLM-as-a-Judge baseline achieves relatively low accuracy on erroneous trajectories, with performance degrading on harder benchmarks. Our unsupervised PRM substantially improves error detection, with the largest gains on the most challenging datasets: +15% on the Omni-MATH dataset and +13% on the OlympiadBench dataset.

Table D4: Full results on the ProcessBench dataset. We report accuracy on erroneous trajectories (Err.), accuracy on correct trajectories (Corr.), and their aggregation via F1 score.

| | **PRM800K** | | | **GSM8K** | | | **MATH** | | | **OlympiadBench** | | | **Omni-MATH** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Err. | Corr. | F1 | Err. | Corr. | F1 | Err. | Corr. | F1 | Err. | Corr. | F1 | Err. | Corr. | F1 |
| LLM-as-a-Judge | 0.25 | 0.57 | 0.34 | 0.37 | 0.75 | 0.50 | 0.33 | 0.61 | 0.43 | 0.22 | 0.46 | 0.29 | 0.19 | 0.44 | 0.27 |
| uPRM (ours) | **0.33** | **0.65** | **0.43** | **0.44** | **0.89** | **0.58** | **0.41** | **0.72** | **0.53** | **0.35** | **0.55** | **0.43** | **0.34** | **0.48** | **0.40** |
| Improvement | +8% | +8% | +9% | +7% | +14% | +8% | +8% | +11% | +10% | +13% | +9% | +14% | +15% | +4% | +13% |

### D.8 Comparison of different score aggregation in TTS with uPRM

Test-time scaling (TTS) requires a score for each candidate answer, which is used to guide and select the final response. These scores are provided with a reward model. To compute the score for a candidate answer, we need to define an aggregation function for the step-wise rewards assigned by a PRM. Common aggregation functions include *(i) last*, where the reward for the last step is assigned to the entire answer, *(ii) product*, where the product of all step-level scores is used, and *(iii) min*, where the minimum step-level reward is the final reward. Multiple studies have investigated the impact of these aggregation methods (Lightman et al., 2024; Snell et al., 2025; Zhang et al., 2025) and have found that this impact is PRM-dependent. Further, Lightman et al. (2024) shows that the difference between *min* and *product* is minor. Therefore, we perform ablation with two strategies, *last* and *product*, using the Best-of-8 sampling strategy, and observe that *last* marginally outperforms *product* in our uPRM; thus, we use *last* aggregation for all TTS experiments in Section 5.2. We adopt `search-and-learn` codebase[4] for our test-time scaling experiments.

Table D5: Performance comparison between *last* and *product* aggregation in the Best-of-8 strategy across different LLMs with our uPRM.

| Policy | Agg. | MATH 500 | Minerva Math | Olympiad Bench | Avg. |
|---|---|---|---|---|---|
| Qwen2.5-1.5B-Instruct | last | $67.1_{\pm 0.6}$ | $26.7_{\pm 1.4}$ | $28.3_{\pm 0.4}$ | **40.7** |
| | product | $68.1_{\pm 0.6}$ | $25.6_{\pm 1.1}$ | $28.0_{\pm 0.6}$ | 40.6 |
| Qwen2.5-7B-Instruct | last | $81.1_{\pm 0.1}$ | $47.2_{\pm 0.8}$ | $44.6_{\pm 0.4}$ | 57.6 |
| | product | $81.1_{\pm 0.6}$ | $47.5_{\pm 1.1}$ | $44.9_{\pm 0.8}$ | **57.8** |
| Llama-3.2-1B-Instruct | last | $46.6_{\pm 0.7}$ | $14.0_{\pm 1.6}$ | $13.4_{\pm 1.5}$ | **24.7** |
| | product | $45.3_{\pm 0.9}$ | $12.7_{\pm 1.3}$ | $13.1_{\pm 0.9}$ | 23.7 |
| Llama-3.1-8B-Instruct | last | $64.5_{\pm 0.6}$ | $34.8_{\pm 1.1}$ | $27.1_{\pm 1.2}$ | **42.1** |
| | product | $63.5_{\pm 1.8}$ | $35.5_{\pm 1.5}$ | $27.2_{\pm 0.3}$ | 42.1 |

### D.9 uPRM initialized from Llama-3-70B

In this section, we show performance for uPRM initialized from Llama-3-70B (Grattafiori et al., 2024) on scaling test-time compute. Note that this model belongs to a different family than the one used in Section 5.2, where uPRM was initialized from Qwen-2.5.14B-Instruct. We fix the sampling strategy to Best-of-N and vary the number of generations from 2 to 128, increasing in powers of 2. The results are available in Table D6. Across all three datasets, we observe that our uPRM, initialized from a different family of instruction-tuned model (Llama), also improves LLMs' performance with scaling test-time compute, showing that our unsupervised training algorithm works across families.

[4] `https://github.com/huggingface/search-and-learn`

Table D6: Best-of-$N$ accuracy of uPRM initialized from Llama-3-70B when scaling test-time compute from 2 to 128 generations across two policy LLMs and three datasets.

| Dataset | Policy | $N=2$ | $N=4$ | $N=8$ | $N=16$ | $N=32$ | $N=64$ | $N=128$ |
|---|---|---|---|---|---|---|---|---|
| MATH-500 | Llama-3.2-1B | $36.3_{\pm 1.9}$ | $41.7_{\pm 1.0}$ | $45.3_{\pm 0.8}$ | $46.7_{\pm 0.8}$ | $48.5_{\pm 0.7}$ | $48.5_{\pm 1.1}$ | $50.5_{\pm 0.1}$ |
| | Qwen2.5-1.5B | $60.5_{\pm 1.3}$ | $64.5_{\pm 0.8}$ | $67.5_{\pm 1.2}$ | $70.1_{\pm 0.8}$ | $70.6_{\pm 0.7}$ | $71.5_{\pm 1.5}$ | $72.2_{\pm 0.7}$ |
| Minerva Math | Llama-3.2-1B | $8.8_{\pm 0.6}$ | $10.8_{\pm 0.6}$ | $12.8_{\pm 1.9}$ | $13.7_{\pm 1.7}$ | $15.4_{\pm 1.7}$ | $15.9_{\pm 1.7}$ | $15.8_{\pm 1.0}$ |
| | Qwen2.5-1.5B | $23.3_{\pm 1.2}$ | $25.5_{\pm 1.1}$ | $27.6_{\pm 1.3}$ | $28.6_{\pm 1.3}$ | $30.0_{\pm 0.2}$ | $30.8_{\pm 0.8}$ | $31.4_{\pm 0.8}$ |
| OlympiadBench | Llama-3.2-1B | $9.1_{\pm 0.6}$ | $11.2_{\pm 1.1}$ | $13.2_{\pm 0.1}$ | $13.8_{\pm 0.6}$ | $15.7_{\pm 0.3}$ | $16.5_{\pm 1.1}$ | $17.2_{\pm 1.0}$ |
| | Qwen2.5-1.5B | $24.2_{\pm 0.8}$ | $26.7_{\pm 1.6}$ | $28.1_{\pm 0.4}$ | $29.6_{\pm 0.6}$ | $30.7_{\pm 0.3}$ | $31.6_{\pm 0.5}$ | $31.4_{\pm 0.8}$ |

### D.10 Reinforcement Learning with PRM

In this section, we provide additional analysis and results obtained in our RL experiments with PRM as a reward source.

#### D.10.1 Reward Hacking Analysis

As discussed in the main text, although reward hacking can occur during training with both uPRM and sPRM, it manifests differently for the two PRMs.

In Figure D2, we report both average length of the generated responses and KL divergence to the reference policy for Qwen2.5-7B trained with uPRM or sPRM rewards only. One can notice that the uPRM-trained model not only experiences RH substantially later, but also stays closer to the reference policy and produces lengthier responses.

Deeper analysis reveals that RH induced by sPRM can be attributed to Case 3 (0 steps) according to Cheng et al. (2025), *i.e.*, the policy learns to output empty or nonsensical responses that are nonetheless highly awarded by the PRM. Meanwhile, training Qwen2.5-7B with uPRM eventually results in Case 2 (1 step) RH, *i.e.*, the policy outputs a single reasoning step and stops generation. As an example, consider the following input prompt: *"What is the sum of the value(s) of $n$ for which $|2n-7|=3$? Please reason step by step with steps separated by "\n\n" and put your final answer within \boxed{}"*. After the policy trained with sPRM collapses around iteration 30, as evidenced by a sharp drop in the response length and increase in the KL divergence, it starts producing empty or extra short responses (*e.g.*, *"\n\n"*), completely neglecting the asked question. In contrast, RH of uPRM, which happens around the 100th training iteration, also results in a significantly reduced response length, but the policy output remains sensible: *"To solve the equation $|2n-7|=3$, we need to consider the definition of the absolute value function, which leads to two possible cases: $2n-7=3$ and $2n-7=-3$"*. We put further discussion as well as additional plots and examples in Appendix D.10.2.

It is of interest to characterize the mechanisms underlying robustness of uPRM to trivial RH. Since uPRM and sPRM differ only in their training procedure, while other factors like dataset or model type are fixed, the observed effect must stem from our method. There are two potential causes: *(i)* the favorable effects of unsupervised learning that prevent PRM from overfitting on specific labeling patterns in the data (Alazraki et al., 2025) or *(ii)* an implicit bias in the learning method itself, *i.e.*, RL vs. SFT (Chu et al., 2025; Shenfeld et al., 2025). To distinguish between these possibilities, we trained an additional PRM with SFT on the PRM800K dataset, replacing the ground-truth labels with per-step labels generated by uPRM. This ablation, therefore, varies one axis at a time: it preserves the SFT training procedure used for sPRM while replacing the labeling pattern with that learned by uPRM. We then used the resulting uPRM-SFT as the reward model for RL training of Qwen2.5-7B and found that trivial RH disappeared. The policy still hacked the reward, but instead of producing trivial responses, it generated a single long reasoning step containing the full solution (see example in Figure D3). Such behavior closely matches Case 2 RH in the taxonomy of Cheng et al. (2025), and differs conceptually from the Case 3 RH observed with sPRM. These results suggest that susceptibility to different types of RH is inherited primarily from the data-labeling pattern rather than from the training procedure itself.

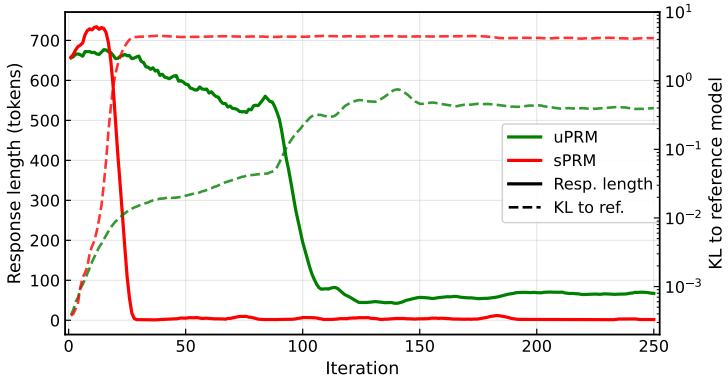



Figure D2: Average response length in tokens and KL divergence to the reference policy for Qwen2.5-7B trained with uPRM and sPRM as the reward source, respectively.

uPRM-SFT Generation Example

**Prompt:** What is the largest whole number value of $n$ that makes the following inequality true? $\frac{1}{3} + \frac{n}{7} < 1$. Please reason step by step with steps separated by "\n\n" and put your final answer within \boxed{}.

**Completion:** We start with the given inequality: $\frac{1}{3} + \frac{n}{7} < 1$. First, we aim to isolate the term involving $n$. To do this, we subtract $\frac{1}{3}$ from both sides of the inequality: $\frac{n}{7} < 1 - \frac{1}{3}$. Next, we need to simplify the right-hand side. We express $1$ as a fraction with a denominator of $3$: $1 = \frac{3}{3}$. Thus, we have $1 - \frac{1}{3} = \frac{3}{3} - \frac{1}{3} = \frac{2}{3}$. The inequality now becomes $\frac{n}{7} < \frac{2}{3}$. To eliminate the fractions, we cross-multiply. This step is valid because both $\frac{n}{7}$ and $\frac{2}{3}$ are positive (assuming $n$ is a positive integer): $n \cdot 3 < 2 \cdot 7$. Simplifying both sides, we get $3n < 14$. Next, we solve for $n$ by dividing both sides of the inequality by $3$: $n < \frac{14}{3}$. We then convert $\frac{14}{3}$ to a decimal to better understand the value: $\frac{14}{3} \approx 4.6667$. This tells us that $n$ must be less than $4.6667$. Since $n$ is a whole number, the largest possible value for $n$ is $4$. To confirm that $n = 4$ satisfies the original inequality, we substitute $n = 4$ back into the inequality: $\frac{1}{3} + \frac{4}{7} < 1$. We find a common denominator for the fractions $\frac{1}{3}$ and $\frac{4}{7}$. The least common multiple of $3$ and $7$ is $21$. Thus, we rewrite the fractions with a denominator of $21$: $\frac{1}{3} = \frac{7}{21}$ and $\frac{4}{7} = \frac{12}{21}$. Adding these fractions, we get $\frac{7}{21} + \frac{12}{21} = \frac{19}{21}$. Therefore, the inequality becomes $\frac{19}{21} < 1$, which is true. Consequently, $n = 4$ indeed satisfies the original inequality. To ensure that $n = 4$ is the largest possible value, we check $n = 5$. Substituting $n = 5$ into the inequality, we get $\frac{1}{3} + \frac{5}{7} < 1$. Again, we find a common denominator of $21$ for the fractions $\frac{1}{3}$ and $\frac{5}{7}$: $\frac{1}{3} = \frac{7}{21}$ and $\frac{5}{7} = \frac{15}{21}$. Adding these fractions, we obtain $\frac{7}{21} + \frac{15}{21} = \frac{22}{21}$. Therefore, the inequality becomes $\frac{22}{21} < 1$, which is false. Thus, $n = 5$ does not satisfy the original inequality. Consequently, the largest whole number value of $n$ that makes the inequality true is indeed $4$. Therefore, the final answer is $\boxed{4}$.

Figure D3: When training with uPRM-SFT, policy learns to hack the reward by generating the complete solution in a single step.

### D.10.2 Further Analysis and More Results

Figures D4, D5, and D6 depict panels of metrics for all three RL-training runs (with different random seeds) of Qwen2.5 models. As stated in the main text, we consider two process reward models, uPRM

and sPRM, and the following reward options: *(i)* only VR, *(ii)* only PRM rewards, *(iii)* PRM rewards + VR on 10% of the data. We plot the following metrics (averaged over responses in the batch):

1. **Accumulated PRM reward.** Mathematically, this is the PURE return value for the first step in the response, computed according to equation (6) in Cheng et al. (2025); effectively, it approximates the minimum of per-step PRM-emitted rewards for a given response. By definition, verifiable reward is not taken into account when computing this value, therefore we do not plot it for the VR run. Since uPRM and sPRM yield different reward models, it should be noted that their accumulated rewards are not directly comparable.
2. **Response length.** Amount of tokens in the response generated by the model for a given input prompt.
3. **KL to reference model.** Kullback–Leibler divergence between the current policy and the reference policy computed over response tokens. Reference policy is defined by the model at initialization (zero-shot policy).

**Analysis.** As evidenced from the plots, Qwen2.5-Math models could be successfully trained using uPRM both with and without VR. Training with sPRM, even in combination with VR, resulted in reward hacking for all considered models. This can be noticed by sharp transitions in all metrics: sudden increase in the reward and KL and drop in the response length. We found that, after RH occurs, each model trained with sPRM (+ VR) converges to the same degenerate behavior of producing empty or extremely short responses, like *"\n\n"*, which is highly rewarded by sPRM ($\sim 0.65$ accumulated reward).

During RL training of Qwen2.5-7B, both uPRM and sPRM were exposed to RH. However, as discussed earlier, the types of their RH differ substantially: training with sPRM results in the most trivial Case 3 (0 step) RH according to Cheng et al. (2025), while RH in uPRM can be attributed to Case 2 (1 step).

It can be observed that occasionally, when training with the sPRM reward, another RH transition occurs at a later stage of training: note, *e.g.*, a drop in KL around iteration 300 for sPRM-trained Qwen2.5-7B in Figure D4a (solid red line). Still, the type of RH is not changed essentially. The model switches from giving empty-like responses to trivially continuing the input prompt, for example, as follows: *"You do not need to box your final answer if it is a variable or an expression"* or *"If you need to use variables or special functions, define them clearly"*.[5]

At the same time, reward hacking that happens in training Qwen2.5-7B with uPRM or uPRM + VR is milder. The hacking behavior does not change throughout training and consists in outputting a single but correct reasoning step, which consequentially receives a high reward from the PRM. Some examples are provided in Table D7.

Interestingly, combining PRM rewards with VR neither necessarily improves final performance nor delays the onset of reward hacking. For instance, for Qwen2.5-Math-1.5B in Figure D4c, RH occurred *earlier* with sPRM + VR than with sPRM. Potentially, changing the weights before reward terms or increasing the proportion of data for which VR is calculated could have an impact.

### D.10.3 Extended results with additional baselines

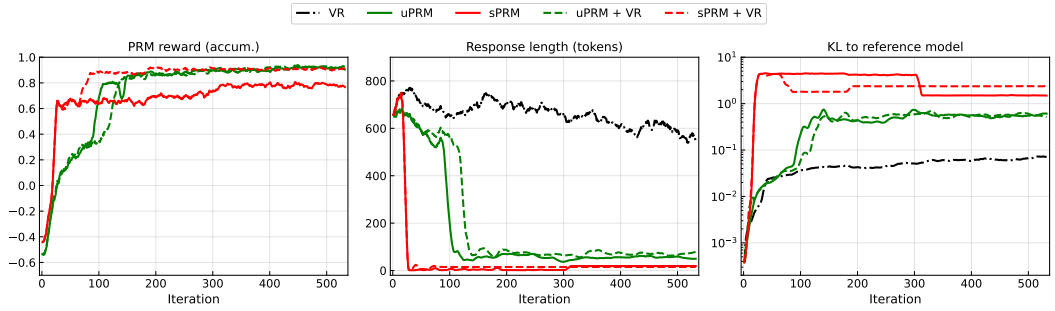
We additionally evaluate the baselines considered in Section 5.1: LLM-as-a-Judge used as a process reward model and a PRM trained on the CoT-labeled PRM800K dataset (CoT-PRM).

Table D8 extends the main results from Table 2 with these additional reward models, allowing all methods to be compared directly. Across all three policy models and mathematical benchmarks, both LLM-as-a-Judge and CoT-PRM underperform the corresponding uPRM variants, including when combined with VR. Notably, both baselines are unsupervised in the sense that they do not require human-provided process labels, further demonstrating the advantage of uPRM over alternative unsupervised PRM approaches.

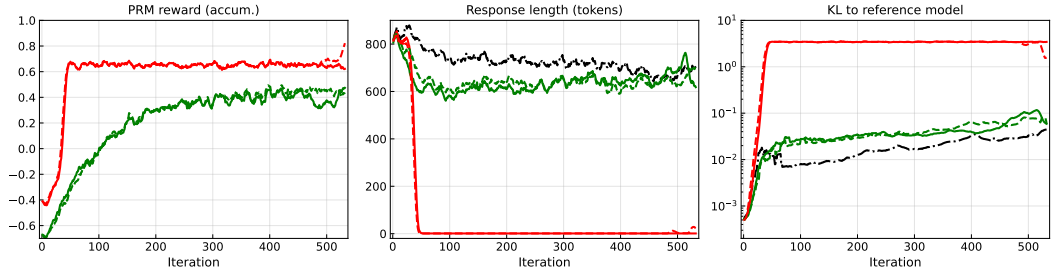
[5] Model inputs are created by appending a question from the dataset with the standard instruction: *"Please reason step by step with steps separated by "\n\n" and put your final answer within \boxed{}"*.

Table D7: **Qwen2.5-7B post-reward hacking examples for training with uPRM rewards.** Sample generations with high PRM reward that exhibit "solution-introduction" behavior: beginning a correct derivation but not completing it.

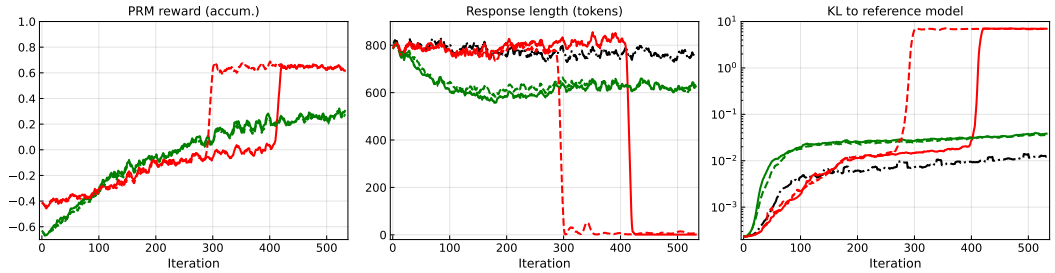
| Iter. | Question | Model generation | PRM reward |
|---|---|---|---|
| 100 | A parabola with equation $y = x^2 + bx + c$ passes through $(-1, -11)$ and $(3, 17)$. What is $c$? | "To find the value of $c$ in the equation $y = x^2 + bx + c$, we start by using the given points $(-1, -11)$ and $(3, 17)$ that lie on the parabola." | 0.98 |
| 150 | What is the sum of the value(s) of $n$ for which $\|2n - 7\| = 3$? | "To solve the equation $\|2n - 7\| = 3$, we need to consider the definition of the absolute value function, which leads to two possible cases: $2n - 7 = 3$ and $2n - 7 = -3$." | 0.99 |
| 200 | Find the least integer value of $x$ for which $2\|x\| + 7 < 17$. | "To find the least integer value of $x$, we start by isolating the absolute value term. We begin with the inequality: $2\|x\| + 7 < 17$." | 0.97 |
| 300 | Using side lengths $2, 3, 5, 7, 11$, how many triangles with exactly two equal sides can be formed? | "To form a triangle, the triangle inequality must be satisfied, which states that the sum of the lengths of any two sides must be greater than the length of the remaining side." | 0.94 |
| 400 | For how many $n \in \{2, 3, \ldots, 100\}$ is $235236_n$ a multiple of 7? | "We start by expressing the base-$n$ number $235236_n$ in base-10. The number $235236_n$ can be expanded as $2n^5+3n^4+5n^3+2n^2+3n+6$." | 0.99 |
| 500 | Solve for $c$: $\frac{c-23}{2} = \frac{2c+5}{7}$. | "We start by eliminating the denominators through cross-multiplication. The given equation is $\frac{c-23}{2} = \frac{2c+5}{7}$." | 0.98 |




(a) **Qwen2.5-7B**

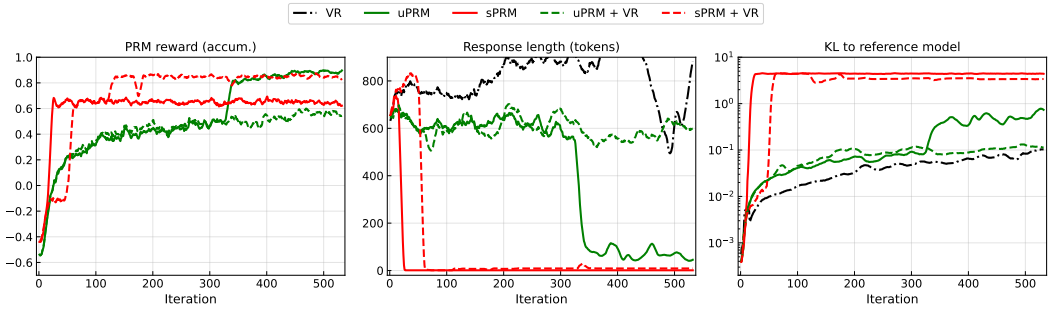



(b) **Qwen2.5-Math-7B**

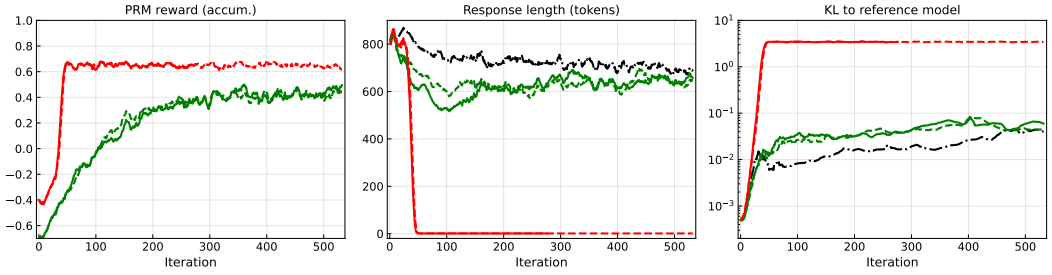



(c) **Qwen2.5-Math-1.5B**

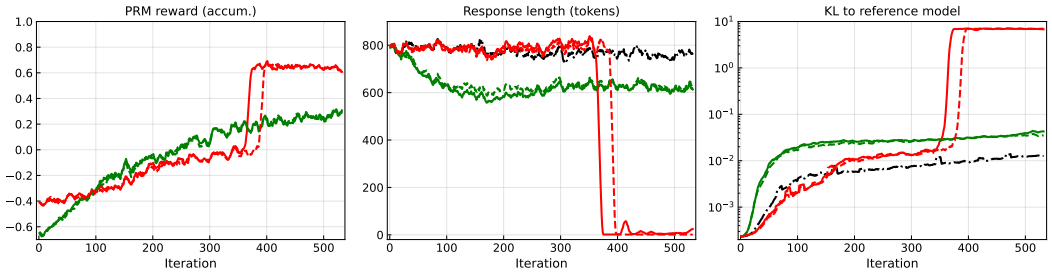
Figure D4: **Training run (seed 1).** Accumulated PRM reward, response length in tokens, and KL divergence to the reference policy for Qwen2.5 models trained with different reward sources. Accumulated reward is computed according to eq. (6) in Cheng et al. (2025), and represents an approximate minimum of per-step PRM rewards for a given response (VR is not taken into account). Note that formally uPRM and sPRM reward values are incomparable due to different reward models.




(a) **Qwen2.5-7B**

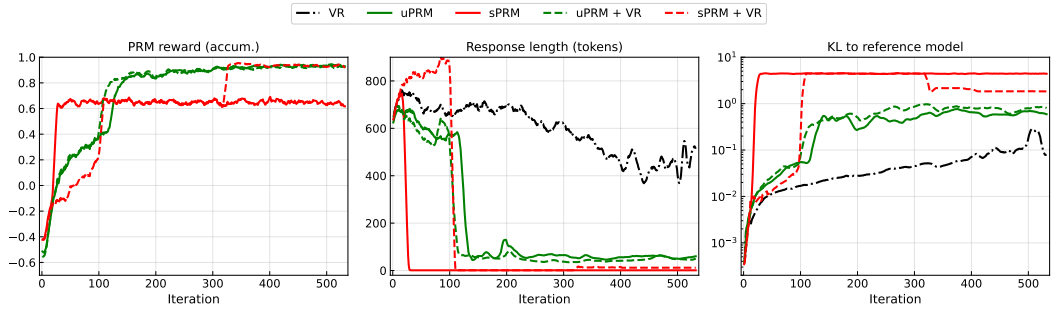



(b) **Qwen2.5-Math-7B**

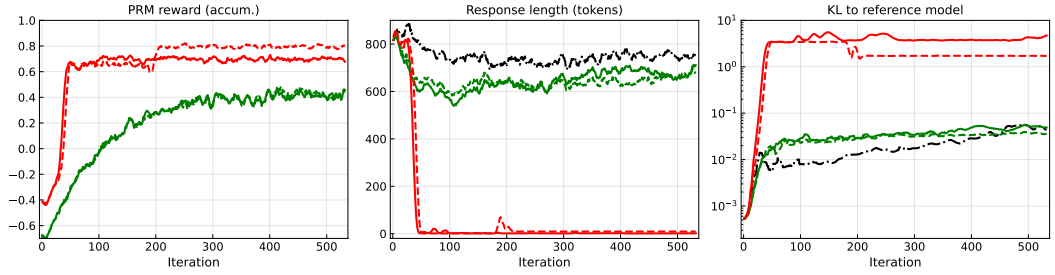



(c) **Qwen2.5-Math-1.5B**

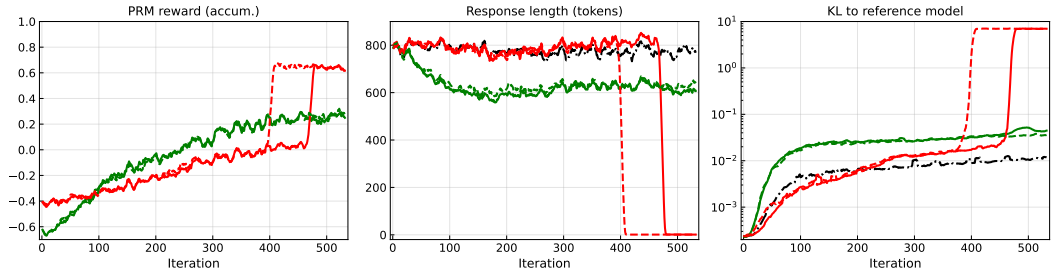
Figure D5: **Training run (seed 2).** Accumulated PRM reward, response length in tokens, and KL divergence to the reference policy for Qwen2.5 models trained with different reward sources. Accumulated reward is computed according to eq. (6) in Cheng et al. (2025), and represents an approximate minimum of per-step PRM rewards for a given response (VR is not taken into account). Note that formally uPRM and sPRM reward values are incomparable due to different reward models.




(a) **Qwen2.5-7B**




(b) **Qwen2.5-Math-7B**




(c) **Qwen2.5-Math-1.5B**

Figure D6: **Training run (seed 3).** Accumulated PRM reward, response length in tokens, and KL divergence to the reference policy for Qwen2.5 models trained with different reward sources. Accumulated reward is computed according to eq. (6) in Cheng et al. (2025), and represents an approximate minimum of per-step PRM rewards for a given response (VR is not taken into account). Note that formally uPRM and sPRM reward values are incomparable due to different reward models.

Table D8: Extended accuracy results on mathematical benchmarks after RL training with different reward sources. This table includes all reward sources reported in Table 2 in the main text, together with the additional LLM-as-a-Judge and CoT-PRM baselines. CoT-PRM denotes a process reward model trained on chain-of-thought critic labels. Entries report mean ± sample standard deviation across seeds. Cells shaded in green correspond to runs with our uPRM. Rows marked with † were evaluated at the last checkpoint before reward hacking, *i.e.*, prior to the end of training for at least one seed. For Qwen2.5-Math-7B, sPRM and sPRM+VR, and for Qwen2.5-7B, sPRM alone could not be evaluated due to rapid reward hacking. **Bold** indicates the best mean score across all reward sources for a given policy model and benchmark.

| Model | Reward | MATH-500 | Minerva Math | OlympiadBench |
|---|---|---|---|---|
| Qwen2.5-7B | VR | $74.1_{\pm 0.8}$ | $34.2_{\pm 1.0}$ | $34.8_{\pm 1.0}$ |
| | sPRM + VR† | $\mathbf{75.4}_{\pm 0.0}$ | $29.4_{\pm 4.7}$ | $36.9_{\pm 0.4}$ |
| | uPRM† | $73.2_{\pm 0.4}$ | $35.0_{\pm 1.3}$ | $\mathbf{37.5}_{\pm 1.1}$ |
| | uPRM + VR† | $73.2_{\pm 1.4}$ | $\mathbf{35.8}_{\pm 0.6}$ | $35.7_{\pm 1.9}$ |
| | CoT-PRM† | $65.3_{\pm 10.9}$ | $31.0_{\pm 1.3}$ | $32.7_{\pm 6.3}$ |
| | CoT-PRM + VR† | $71.9_{\pm 1.6}$ | $32.7_{\pm 1.4}$ | $35.4_{\pm 1.0}$ |
| | LLM-as-a-Judge | $61.7_{\pm 12.9}$ | $24.0_{\pm 2.8}$ | $28.7_{\pm 9.7}$ |
| | LLM-as-a-Judge + VR | $69.2_{\pm 1.8}$ | $27.8_{\pm 4.0}$ | $32.7_{\pm 1.9}$ |
| Qwen2.5-Math-7B | VR | $80.1_{\pm 0.8}$ | $35.9_{\pm 0.4}$ | $41.8_{\pm 0.4}$ |
| | uPRM | $\mathbf{82.9}_{\pm 0.4}$ | $\mathbf{37.9}_{\pm 1.0}$ | $42.1_{\pm 1.3}$ |
| | uPRM + VR | $82.1_{\pm 1.3}$ | $36.3_{\pm 2.2}$ | $\mathbf{43.8}_{\pm 0.2}$ |
| | CoT-PRM† | $71.9_{\pm 0.5}$ | $35.3_{\pm 0.4}$ | $38.2_{\pm 0.4}$ |
| | CoT-PRM + VR† | $73.1_{\pm 0.8}$ | $32.5_{\pm 0.6}$ | $37.7_{\pm 1.2}$ |
| | LLM-as-a-Judge† | $37.8_{\pm 25.6}$ | $15.7_{\pm 10.5}$ | $16.5_{\pm 10.9}$ |
| | LLM-as-a-Judge + VR | $57.6_{\pm 20.7}$ | $26.5_{\pm 13.6}$ | $27.6_{\pm 13.8}$ |
| Qwen2.5-Math-1.5B | VR | $70.0_{\pm 0.4}$ | $26.0_{\pm 0.2}$ | $33.5_{\pm 1.0}$ |
| | sPRM† | $\mathbf{74.7}_{\pm 0.3}$ | $27.8_{\pm 0.6}$ | $35.0_{\pm 1.0}$ |
| | sPRM + VR† | $74.4_{\pm 1.2}$ | $28.7_{\pm 1.3}$ | $36.3_{\pm 0.6}$ |
| | uPRM | $73.5_{\pm 1.2}$ | $\mathbf{31.8}_{\pm 0.8}$ | $\mathbf{36.6}_{\pm 0.6}$ |
| | uPRM + VR | $74.3_{\pm 0.5}$ | $31.5_{\pm 0.8}$ | $35.8_{\pm 0.5}$ |
| | CoT-PRM† | $67.6_{\pm 7.1}$ | $25.6_{\pm 2.8}$ | $31.7_{\pm 3.3}$ |
| | CoT-PRM + VR† | $63.3_{\pm 0.9}$ | $23.0_{\pm 1.1}$ | $29.3_{\pm 0.2}$ |
| | LLM-as-a-Judge | $61.5_{\pm 1.2}$ | $19.6_{\pm 2.5}$ | $25.6_{\pm 1.5}$ |
| | LLM-as-a-Judge + VR | $62.7_{\pm 1.7}$ | $24.3_{\pm 1.3}$ | $25.9_{\pm 0.6}$ |