# LEARNING PROCESS REWARDS VIA REASONING STATE PROPAGATION

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

## ABSTRACT

Process reward models (PRMs) have demonstrated notable effectiveness in test-time scaling and reinforcement learning by providing fine-grained signals for evaluating intermediate reasoning states, but their training relies heavily on costly process annotations. A natural way to alleviate this dependence is to complement limited process supervision with scalable outcome supervision. However, existing PRMs often model reasoning prefixes independently, providing no explicit mechanism for effectively using final outcome to guide the learning of intermediate reasoning states. We introduce *Reasoning State Propagation (*RSP*)*, which represents each reasoning prefix with a binary validity state and models transitions between successive states across the reasoning trajectory. Specifically, RSP predicts a *break* probability that a valid state becomes invalid and a *repair* probability that an invalid state returns to valid. By propagating these transitions, RSP connects intermediate states to the final state, allowing process annotations to supervise intermediate states while outcome labels supervise the final state and can provide learning signals to preceding steps. Across reasoning search, response selection, and reinforcement learning, RSP consistently outperforms representative PRM baselines, with average improvements over Qwen2.5-Math-PRM of 5.6% in beam search and 2.1% in reinforcement learning.

## 1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) (Jaech et al., 2024; Guo et al., 2025; Team et al., 2025; Yu et al., 2026) has driven substantial progress in large reasoning models (LRMs) (Yang et al., 2025a; Seed et al., 2025; Xiaomi et al., 2025). However, RLVR typically relies on final-answer correctness as a trajectory-level reward, reducing a multi-step reasoning process to a single scalar signal. Such coarse supervision cannot distinguish the contributions of individual reasoning steps, making fine-grained credit assignment difficult. This limitation is especially problematic for all-incorrect groups on challenging problems. In group-relative methods such as GRPO (Shao et al., 2024), identical rewards within all-incorrect groups result in zero advantages after group normalization, leaving no learning signal for policy optimization. To address these limitations, PRMs (Lightman et al., 2024; Wang et al., 2024; Luo et al., 2024; Zhang et al., 2024; Li & Li, 2025; Zhang et al., 2025b; Yang et al., 2025b; Zhang et al., 2025a) have emerged to provide step-level feedback throughout the reasoning process.

Despite the notable success of PRMs in test-time scaling (Lightman et al., 2024; Lu et al., 2024; Snell et al., 2025) and reinforcement learning (Li et al., 2026; Agrawal et al., 2026), PRM training typically relies on process annotations, which are costly and difficult to obtain at scale. Such annotations provide supervision over intermediate reasoning steps and are particularly valuable for fine-grained process evaluation. In contrast, outcome labels are substantially cheaper and readily scalable in verifiable domains since they only require determining whether the complete reasoning trajectory ultimately reaches a correct answer. This disparity suggests a natural complementary supervision strategy: scarce process annotations provide guidance on intermediate reasoning states, while abundant outcome labels offer scalable supervision over a much larger collection of trajectories. However, how to effectively integrate these two signals within a PRM remains underexplored.

A key obstacle to effectively exploiting this complementary supervision is how to relate intermediate reasoning states to the final outcome. Existing discriminative PRMs optimize individual prefixes as



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.

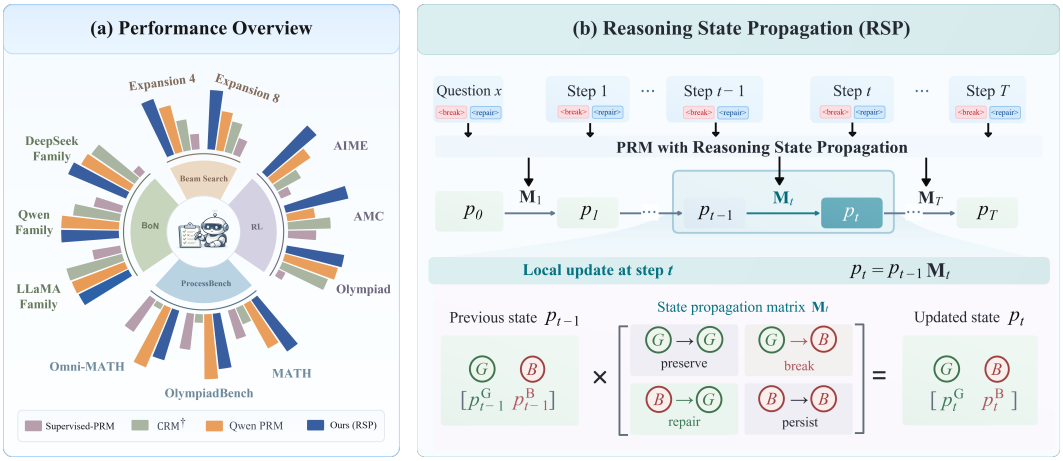



Figure 1: (a) Empirical performance across evaluation settings. (b) RSP propagates a binary reasoning state across steps using *break* and *repair* probabilities. G denotes a *Good or Valid* state with no unresolved error, while B denotes a *Bad or Invalid* state with an unresolved error. Transitions $\mathrm{G} \to \mathrm{B}$ and $\mathrm{B} \to \mathrm{G}$ correspond to error introduction (*break*) and error recovery (*repair*), respectively.

independent classification targets (Lightman et al., 2024; Zhang et al., 2025b), providing no explicit structure for connecting reasoning states across steps. Recent methods such as PQM (Li & Li, 2025) and CRM (Zhang et al., 2025a) introduce sequential dependencies, but largely organize the reasoning process around the first erroneous step and effectively treat subsequent reasoning as remaining invalid. Such a formulation oversimplifies the relationship between intermediate reasoning states and final correctness. An intermediate error may be corrected by later reasoning and therefore does not necessarily lead to an incorrect final outcome. Similarly, a correct final outcome does not imply that every preceding reasoning step is valid. Another limitation is reflected in the popular process annotation protocol of PRM800K (Lightman et al., 2024), where annotation terminates once the first error is identified, leaving subsequent reasoning unannotated. In fact, modern LRMs can detect and revise earlier mistakes during continued reasoning (Guo et al., 2025; Lee et al., 2026). This motivates a more general formulation that explicitly propagates reasoning states across steps.

In this paper, we introduce **Reasoning State Propagation (RSP)** for PRM training. Rather than independently predicting the validity of each reasoning prefix, RSP associates each prefix with an effective reasoning state that indicates whether the currently maintained derivation is valid or contains an unresolved error and propagates this state across successive prefixes. Consider a reasoning trajectory composed of steps $(s_1, \ldots, s_T)$, where $s_t$ denotes the $t$-th reasoning step. When extending the preceding prefix $\mathbf{s}_{<t}$ with step $s_t$, RSP estimates two conditional update probabilities: a *break* probability, which measures the likelihood that a previously valid state becomes invalid after incorporating $s_t$, and a *repair* probability, which measures the likelihood that the new step resolves or supersedes an earlier error and returns the current reasoning state to a valid state. These probabilities determine how the state of the preceding prefix is updated after incorporating $s_t$. Repeating this process throughout the trajectory explicitly links the reasoning states across steps. The framework of RSP is shown in Figure 1(b).

RSP provides a natural interface for jointly leveraging process and outcome supervision. Process annotations can ground the validity of intermediate reasoning states, while outcome labels supervise the final propagated state of the complete trajectory. Since the final state depends on the sequence of preceding state updates, outcome supervision can supervise the final state and provide learning signals through the intervening state updates. This provides a way for outcome feedback to complement the model's locally grounded validity predictions on trajectories without process labels.

Our main contributions are summarized as follows:

- We propose a new formulation RSP for process reward modeling that models transitions between successive reasoning states through *break* and *repair*, rather than treating each reasoning prefix independently, thus explicitly linking reasoning states across the trajectory.

- RSP jointly leverages scarce process annotations and scalable outcome supervision, with process labels grounding intermediate states and outcome labels constraining the final state. We further analyze how both signals provide training signals for state updates.
- Extensive experiments across test-time scaling and reinforcement learning demonstrate consistent improvements over representative PRMs. Comprehensive analyses further validate the effectiveness of reasoning state propagation and its key design choices.

## 2 Related Work

### 2.1 Enhancing Reasoning in LRMs

Recent studies improve the reasoning capabilities of LRMs through test-time scaling and reinforcement learning. Lightman et al. (2024) use process reward models to rank sampled solutions in Best-of-$N$ selection. Xie et al. (2023) introduce self-evaluation-guided beam search, which prunes intermediate reasoning prefixes during generation. REST-MCTS* (Zhang et al., 2024) further incorporates process rewards into Monte Carlo search and uses the resulting trajectories for self-training. Beyond test-time scaling, DeepSeek-R1 (Guo et al., 2025) demonstrates that reinforcement learning with verifiable outcomes can substantially strengthen reasoning capabilities. PURE (Cheng et al., 2026) introduces min-form credit assignment, using future process rewards to provide denser supervision for intermediate reasoning steps. VeriGate (Agrawal et al., 2026) selectively activates PRM-based token-level supervision when outcome rewards fail to provide group-relative learning signal. These advances increasingly rely on reliable and fine-grained process feedback, motivating accurate and scalable PRMs for evaluating intermediate reasoning states.

### 2.2 Process Reward Modeling

PRMs rely on fine-grained signals about intermediate reasoning steps, which are commonly obtained through human annotation (Uesato et al., 2022; Lightman et al., 2024), rollout-based estimation (Wang et al., 2024; Tan et al., 2025), or automated judging (Zhang et al., 2025b; She et al., 2025). PRM800K (Lightman et al., 2024) collects human judgments of step-level correctness and trains discriminative PRMs to classify each reasoning step as valid or invalid. Math-Shepherd (Wang et al., 2024) reduces annotation costs by estimating step correctness from whether sampled continuations can reach the correct final answer. Zhang et al. (2025b) improve automatically constructed labels through consensus filtering. With these labels, conventional PRMs typically optimize stepwise classification objectives without modeling dependencies between adjacent reasoning prefixes. OVM (Yu et al., 2024) learns the probability that a partial trajectory can eventually reach a correct answer using outcome-annotated rollouts. PQM (Li & Li, 2025) formulates process reward modeling as Q-value ranking, capturing relative relationships among steps within a trajectory. CRM (Zhang et al., 2025a) models the first invalid state and links the process rewards to the final outcome using the probability chain rule. Other recent studies improve process reward modeling through dynamic evaluation criteria or retrieval-augmented verification (Yin et al., 2025; Zhu et al., 2025). However, existing PRMs do not jointly exploit process- and outcome-level supervision and explicitly model flexible transitions between reasoning states.

## 3 Method

In this section, we present RSP, including its reasoning state propagation formulation, model parameterization, and learning objectives.

### 3.1 Preliminaries

**Task Definition** Given a question $x$, an LRM policy $\pi$ generates a reasoning trajectory $\mathbf{s} = (s_1, \ldots, s_T)$ autoregressively, where $s_t \sim \pi(\cdot \mid x, \mathbf{s}_{<t})$ and $\mathbf{s}_{<t} = (s_1, \ldots, s_{t-1})$ denotes the preceding reasoning prefix. We consider two types of supervision. For process-annotated trajectories, binary process labels $y_t^{\text{step}} \in \{0, 1\}$ are available for each step, where $y_t^{\text{step}} = 1$ indicates that the reasoning prefix $\mathbf{s}_{\leq t}$ is valid after incorporating step $s_t$. For outcome-annotated trajectories, only a final label $y^{\text{out}} \in \{0, 1\}$ is available, indicating whether the complete trajectory reaches a correct

answer. Let $N_{\text{step}}$ and $N_{\text{out}}$ denote the numbers of process-annotated and outcome-annotated trajectories. For the $i$-th trajectory, $T_i$ denotes its number of reasoning steps, and $\mathcal{A}_i \subseteq \{1, \ldots, T_i\}$ denotes the set of steps with available process annotations. The two training sets are defined as $\mathcal{D}_{\text{step}} = \left\{\left(x_i, \mathbf{s}_i, \{y_{i,t}^{\text{step}}\}_{t\in\mathcal{A}_i}\right)\right\}_{i=1}^{N_{\text{step}}}$, and $\mathcal{D}_{\text{out}} = \{(x_i, \mathbf{s}_i, y_i^{\text{out}})\}_{i=1}^{N_{\text{out}}}$. For clarity, we omit the trajectory index $i$ when discussing a single trajectory. Our goal is to learn a PRM that estimates the state validity of each reasoning prefix $\mathbf{s}_{\le t}$ while jointly leveraging process and outcome supervision.

**Independent Prefix Classification** A common discriminative PRM formulation (Lightman et al., 2024; Zhang et al., 2025b) treats each reasoning prefix as an independent binary classification target. For the $i$-th trajectory, a PRM with parameters $\theta$ predicts $q_{i,t} = p_\theta\left(y_{i,t}^{\text{step}} = 1 \mid x_i, \mathbf{s}_{i,\le t}\right)$, where $q_{i,t}$ denotes the predicted validity of the reasoning prefix after step $s_{i,t}$. The model is trained on the available process annotations using the binary cross-entropy objective

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{\sum_{i=1}^{N_{\text{step}}} |\mathcal{A}_i|} \sum_{i=1}^{N_{\text{step}}} \sum_{t\in\mathcal{A}_i} \left[y_{i,t}^{\text{step}} \log q_{i,t} + \left(1 - y_{i,t}^{\text{step}}\right) \log\left(1 - q_{i,t}\right)\right]. \tag{1}$$

Although each prediction is conditioned on the full reasoning prefix, the objective decomposes over individual prefixes and imposes no explicit relation between their predicted validity across successive steps. Moreover, this objective relies exclusively on process annotations and cannot naturally incorporate outcome-only trajectories.

### 3.2 Reasoning State Propagation

To explicitly relate reasoning states across successive steps, we represent the state of each reasoning prefix as $z_t \in \mathcal{Z} = \{\mathrm{G}, \mathrm{B}\}$. Here, G denotes that the reasoning currently maintained after step $s_t$ is valid and contains no unresolved error, whereas B denotes that an error remains unresolved in the current reasoning state. For each reasoning step $s_t$, we parameterize two probabilities that describe its effect on the preceding reasoning state:

$$\alpha_t \coloneqq p_\theta\left(z_t = \mathrm{B} \mid z_{t-1} = \mathrm{G}, x, \mathbf{s}_{<t}, s_t\right), \tag{2}$$

$$\beta_t \coloneqq p_\theta\left(z_t = \mathrm{G} \mid z_{t-1} = \mathrm{B}, x, \mathbf{s}_{<t}, s_t\right). \tag{3}$$

Here, $\alpha_t$ denotes the *break* probability that step $s_t$ introduces an unresolved error into a previously valid reasoning state, and $\beta_t$ denotes the *repair* probability that step $s_t$ resolves or supersedes an earlier error, returning the current reasoning state to a valid one. The break and repair probabilities define a local state propagation matrix for step $s_t$:

$$\mathbf{M}_t = \begin{bmatrix} 1-\alpha_t & \alpha_t \\ \beta_t & 1-\beta_t \end{bmatrix}. \tag{4}$$

The rows correspond to the preceding state $z_{t-1}$, while the columns correspond to the updated state $z_t$. $\mathbf{M}_t$ describes a one-step propagation between adjacent reasoning states, conditioned on the preceding reasoning prefix and the newly incorporated step $s_t$. Let $\mathbf{p}_t = \begin{bmatrix} p_t^{\mathrm{G}} & p_t^{\mathrm{B}} \end{bmatrix}$ denote the probability distribution over reasoning states after step $s_t$. Before any reasoning step, we initialize the state as $\mathbf{p}_0 = [1, 0]$, and propagate it as

$$\mathbf{p}_t = \mathbf{p}_{t-1}\mathbf{M}_t = \mathbf{p}_0 \prod_{j=1}^{t} \mathbf{M}_j. \tag{5}$$

We use $p_t^{\mathrm{G}}$ as the process score for the reasoning prefix $\mathbf{s}_{\le t}$. Unlike the independently predicted score $q_t$, $p_t^{\mathrm{G}}$ depends on all preceding state updates through sequential propagation. Here, $\mathrm{G} \to \mathrm{G}$ preserves valid reasoning, $\mathrm{G} \to \mathrm{B}$ captures error introduction, $\mathrm{B} \to \mathrm{B}$ captures error persistence, and $\mathrm{B} \to \mathrm{G}$ captures error recovery. The resulting state sequence therefore explicitly links reasoning states across successive steps.

**Transition Parameterization** To estimate the break and repair probabilities, we insert two special tokens, $\langle\mathrm{BREAK}\rangle$ and $\langle\mathrm{REPAIR}\rangle$, at each reasoning boundary. Boundary 0 is placed after the question $x$, and boundary $t$ is placed after reasoning step $s_t$. Specifically, at each boundary $t$, we insert a pair of update tokens $\mathbf{u}_t = (\langle\mathrm{BREAK}\rangle_t, \langle\mathrm{REPAIR}\rangle_t)$, yielding

$$\widetilde{\mathbf{s}} = (x, \mathbf{u}_0, s_1, \mathbf{u}_1, \ldots, s_T, \mathbf{u}_T).$$

Our PRM is built on a pretrained LRM backbone $f_\phi$, which encodes $\widetilde{\mathbf{s}}$ in a single forward pass. Here, $\phi$ denotes the backbone parameters, including the embeddings of the newly introduced special tokens. Let $\mathbf{h}_t^{\mathrm{br}}$ and $\mathbf{h}_t^{\mathrm{rp}}$ denote the hidden representations of $\langle\mathrm{BREAK}\rangle_t$ and $\langle\mathrm{REPAIR}\rangle_t$ at boundary $t$. Since boundary $t$ follows step $s_t$, these representations encode the reasoning prefix $(x, \mathbf{s}_{\le t})$. To characterize the state update induced by step $s_t$, we use the representations at the boundaries before and after $s_t$. The break and repair probabilities are computed as

$$\alpha_t = \sigma\left(g_{\mathrm{br}}\left(\mathbf{h}_{t-1}^{\mathrm{br}} \,\|\, \mathbf{h}_t^{\mathrm{br}}\right)\right), \tag{6}$$

$$\beta_t = \sigma\left(g_{\mathrm{rp}}\left(\mathbf{h}_{t-1}^{\mathrm{rp}} \,\|\, \mathbf{h}_t^{\mathrm{rp}}\right)\right), \tag{7}$$

where $\|$ denotes the vector concatenation. $g_{\mathrm{br}}(\cdot)$ and $g_{\mathrm{rp}}(\cdot)$ are two independent MLP heads, and $\sigma(\cdot)$ denotes the sigmoid function. All break and repair probabilities along the trajectory are obtained from a single forward pass. We denote all trainable parameters of RSP by $\theta$.

### 3.3 Learning from Process and Outcome Supervision

The propagated reasoning states enable RSP to naturally incorporate both process and outcome supervision. We define the corresponding learning objectives below.

For each process-annotated trajectory in $\mathcal{D}_{\mathrm{step}}$, we supervise the propagated state at every annotated step $t \in \mathcal{A}_i$. Since $y_{i,t}^{\mathrm{step}} = 1$ indicates a valid reasoning step, we define

$$\mathcal{L}_{\mathrm{step}} = -\frac{1}{\sum_{i=1}^{N_{\mathrm{step}}} |\mathcal{A}_i|} \sum_{i=1}^{N_{\mathrm{step}}} \sum_{t \in \mathcal{A}_i} \left[ y_{i,t}^{\mathrm{step}} \log p_{i,t}^{\mathrm{G}} + \left(1 - y_{i,t}^{\mathrm{step}}\right) \log p_{i,t}^{\mathrm{B}} \right]. \tag{8}$$

Unlike BCE in equation 1 applied to independently predicted scores $q_{i,t}$, this loss supervises state probabilities that depend on the preceding reasoning states through sequential propagation.

For each outcome-annotated trajectory in $\mathcal{D}_{\mathrm{out}}$, we define

$$\mathcal{L}_{\mathrm{out}} = -\frac{1}{N_{\mathrm{out}}} \sum_{i=1}^{N_{\mathrm{out}}} \left[ y_i^{\mathrm{out}} \log p_{i,T_i}^{\mathrm{G}} + \left(1 - y_i^{\mathrm{out}}\right) \log p_{i,T_i}^{\mathrm{B}} \right]. \tag{9}$$

Because the final state $\mathbf{p}_{i,T_i}$ depends on all state updates along the trajectory, the outcome loss is backpropagated through the propagation process and therefore trains the break and repair predictions at intermediate steps. We jointly optimize the two supervision signals as $\mathcal{L}_{\mathrm{total}} = \mathcal{L}_{\mathrm{step}} + \mathcal{L}_{\mathrm{out}}$.

**Effect of Outcome Supervision on State Transitions** Process labels directly supervise intermediate states, whereas outcome supervision reaches earlier transitions through state propagation. For an outcome-annotated trajectory, let $c \in \{\mathrm{G}, \mathrm{B}\}$ denote the terminal target and define $h_t(a) = p_\theta(z_T = c \mid z_t = a, x, \mathbf{s})$ as the likelihood of reaching $c$ from state $a$ through the remaining reasoning. Let $Z_{\mathrm{out}} = p_T^c$ and $\ell^{\mathrm{out}} = -\log Z_{\mathrm{out}}$. Then, with respect to the pre-sigmoid logits of $\alpha_t$ and $\beta_t$,

$$\frac{\partial \ell^{\mathrm{out}}}{\partial \operatorname{logit}(\alpha_t)} = \frac{p_{t-1}^{\mathrm{G}} \alpha_t (1-\alpha_t)}{Z_{\mathrm{out}}} \big(h_t(\mathrm{G}) - h_t(\mathrm{B})\big), \quad \frac{\partial \ell^{\mathrm{out}}}{\partial \operatorname{logit}(\beta_t)} = \frac{p_{t-1}^{\mathrm{B}} \beta_t (1-\beta_t)}{Z_{\mathrm{out}}} \big(h_t(\mathrm{B}) - h_t(\mathrm{G})\big).$$

These gradients show how the final outcome shapes intermediate state transitions. If $h_t(\mathrm{G}) > h_t(\mathrm{B})$, the outcome loss favors transitions toward G by decreasing *break* and increasing *repair*, with the opposite effect when $h_t(\mathrm{B}) > h_t(\mathrm{G})$. Detailed analysis is provided in Appendix C.

## 4 Experiments

### 4.1 Experimental Settings

**Model Architecture** We build RSP on the Qwen3 family, using Qwen3-4B as the default backbone unless otherwise specified. Following the formulation of RSP, we use two independent MLP heads to predict the *break* and *repair* probabilities. The backbone parameters, the embeddings of the $\langle\mathrm{BREAK}\rangle$ and $\langle\mathrm{REPAIR}\rangle$ tokens, and the two MLP heads are optimized during training.

Table 1: Beam search accuracy (%) on MATH500 and Gaokao. The beam width is fixed to 4, and $e$ denotes the expansion width. The best and second-best results are shown in **bold** and underlined.

| PRM Backbone | Reward Model | MATH500 | | | Gaokao | | |
|---|---|---|---|---|---|---|---|
| | | $e=4$ | $e=8$ | $e=12$ | $e=4$ | $e=8$ | $e=12$ |
| Qwen2.5-Math-7B | Qwen2.5-Math-PRM | 74.8 | 74.0 | 75.8 | 58.5 | 55.6 | 59.3 |
| Qwen3-1.7B | Supervised PRM | 54.8 | 53.6 | 55.4 | 44.0 | 45.7 | 48.3 |
| | OVM | 67.9 | 68.4 | 69.5 | 54.5 | 58.3 | 58.3 |
| | Pseudo-Label PRM | 65.3 | 64.8 | 67.7 | 53.1 | 52.5 | 55.0 |
| | Joint-Supervised PRM | 67.4 | 64.6 | 64.0 | 54.4 | 57.2 | 55.9 |
| | CRM$^\dagger$ | 62.2 | 66.2 | 63.4 | 55.6 | 53.5 | 56.6 |
| | **RSP** | **68.8** | **70.5** | **70.0** | **55.6** | **58.5** | **59.3** |
| Qwen3-4B | Supervised PRM | 62.3 | 59.8 | 61.4 | 52.6 | 53.3 | 52.7 |
| | OVM | 73.1 | 75.2 | 74.9 | 61.7 | 64.8 | 63.0 |
| | Pseudo-Label PRM | 68.2 | 64.4 | 65.7 | 56.4 | 49.9 | 53.4 |
| | Joint-Supervised PRM | 75.4 | 75.4 | 75.6 | 62.9 | 63.2 | 61.1 |
| | CRM$^\dagger$ | 68.5 | 68.2 | 69.9 | 55.3 | 55.3 | 56.8 |
| | **RSP** | **76.0** | **78.6** | **79.8** | **63.5** | **65.8** | **67.9** |

**Training Data** We train RSP using both process and outcome supervision. For process supervision, we use PRM800K (Lightman et al., 2024), which contains human process annotations for trajectories. Unless otherwise specified, we randomly select 20% of PRM800K as the process-annotated training set. For outcome supervision, we use the filtered correctness labels from AceMath-RM (Liu et al., 2025b) as outcome supervision for the terminal reasoning state. AceMath-RM ranks and filters candidate responses with an LLM judge to retain high-confidence examples, reducing cases where final answer correctness is inconsistent with the validity of the terminal reasoning state. Each batch contains process-annotated and outcome-annotated examples at a default ratio of 1:3. We train models for two epochs, where one epoch corresponds to a complete pass over the process-annotated set, and outcome-annotated examples are sampled to maintain the ratio.

**Baselines** We compare RSP with representative process- and outcome-supervised reward models. Supervised PRM (Lightman et al., 2024) independently predicts the validity of each reasoning prefix using process annotations. OVM (Yu et al., 2024) uses outcome supervision by assigning the outcome as the target for each reasoning step. We additionally compare with Qwen2.5-Math-PRM (Zhang et al., 2025b), using its 7B checkpoint by default, and CRM (Zhang et al., 2025a) as another representative PRM. To compare different ways of combining process and outcome supervision, we construct additional baselines: Pseudo-Label PRM, Joint-Supervised PRM, and CRM$^\dagger$. Pseudo-Label PRM follows a standard semi-supervised strategy (Sohn et al., 2020), using high-confidence predictions as pseudo process labels. Joint-Supervised PRM extends Supervised PRM by additionally supervising the prediction at the final reasoning prefix with the outcome label using BCE loss. CRM$^\dagger$ extends CRM with an additional outcome objective, allowing it to incorporate both process and outcome supervision. More details are provided in Appendix A.

### 4.2 Process-Guided Beam Search

We evaluate whether process rewards can reliably rank intermediate reasoning prefixes in beam search. We consider PRMs built on Qwen3-1.7B and Qwen3-4B and evaluate them on MATH500 (Lightman et al., 2024) and Gaokao (Liao et al., 2024). The policy model is fixed to Qwen3-1.7B for all experiments. We fix the beam width to $b = 4$ and vary the expansion width $e \in \{4, 8, 12\}$. At each search step, each of the $b$ retained prefixes is expanded into $e$ candidate continuations. The resulting $b \times e$ candidates are ranked by the PRM, and the top-$b$ prefixes are retained for the next step.

**Analysis** Table 1 shows that RSP achieves the strongest overall beam search performance across both Qwen3 PRM backbones and expansion widths. The advantage is particularly pronounced with

Table 2: Average Best-of-$N$ accuracy (%) over $N \in \{8, 16, 32, 64, 128\}$. Proc./Out. denote process/outcome supervision, and ✓$^+$ indicates additional large-scale process supervision.

| Method | Proc. | Out. | LLaMA3 70B | LLaMA3.3 70B | Qwen3 1.7B | Qwen3 4B | Qwen3 8B | DeepSeek-R1-0528 Qwen3-8B | Avg. |
|---|---|---|---|---|---|---|---|---|---|
| Pass@1 | – | – | 43.6 | 69.0 | 65.8 | 77.4 | 79.2 | 51.6 | 64.4 |
| Supervised PRM | ✓ | – | 57.2 | 74.6 | 76.0 | 80.8 | 79.9 | 46.0 | 69.1 |
| OVM | – | ✓ | 60.5 | 75.5 | 77.9 | <u>84.1</u> | <u>83.2</u> | <u>62.4</u> | 73.9 |
| CRM | ✓ | – | 58.2 | **77.8** | <u>79.1</u> | 83.1 | 82.5 | 52.1 | 72.1 |
| Qwen2.5-Math-PRM | ✓$^+$ | – | 60.6 | 77.1 | **79.9** | 83.5 | 82.4 | 61.0 | <u>74.1</u> |
| Pseudo-Label PRM | ✓ | ✓ | 43.6 | 69.2 | 65.8 | 77.6 | 80.2 | 59.9 | 66.1 |
| Joint-Supervised PRM | ✓ | ✓ | <u>60.7</u> | 77.1 | 78.8 | 82.4 | 82.9 | 59.9 | 73.6 |
| CRM$^\dagger$ | ✓ | ✓ | 60.6 | <u>77.4</u> | 78.6 | 82.0 | 81.9 | 60.0 | 73.4 |
| **RSP** | ✓ | ✓ | **61.2** | 76.1 | 78.3 | **84.1** | **83.3** | **63.3** | **74.4** |

the Qwen3-4B backbone at larger expansion widths. At $e = 12$, RSP outperforms OVM by 4.9% on both MATH500 and Gaokao. OVM remains competitive, suggesting that outcome supervision alone can provide useful signals for prefix ranking. In contrast, Pseudo-Label PRM performs substantially worse and exhibits larger variation across expansion widths, particularly on Gaokao. Consistent gains over Joint-Supervised PRM and CRM$^\dagger$ further show the benefit of reasoning state propagation when jointly leveraging process and outcome supervision.

### 4.3 Best-of-$N$ Selection

Best-of-$N$ evaluates whether a reward model can select a correct response from multiple sampled candidates. For each generator and question, we sample $N$ responses and select the response with the highest final score. We consider $N \in \{8, 16, 32, 64, 128\}$ and report the average accuracy in these settings. All evaluations are conducted on MATH500. For PRMs, we use the process score of the final reasoning step as the response score. We consider six generators from the LLaMA, Qwen, and DeepSeek model families.

**Analysis** Table 2 shows that RSP achieves the highest average Best-of-$N$ accuracy across the six generators. In particular, RSP slightly exceeds Qwen2.5-Math-PRM in average accuracy despite the latter using additional large-scale process supervision. Adding outcome supervision to CRM improves its average accuracy from 72.1% to 73.4%, while RSP further improves it to 74.4%. In contrast, Pseudo-Label PRM performs substantially worse, indicating that pseudo process labels provide a less effective way to combine the two supervision sources. Joint-Supervised PRM is also competitive but remains below RSP, showing that it does not fully exploit the two supervision signals. The results remain competitive across LLaMA, Qwen, and DeepSeek generators, indicating that the benefit of RSP is not limited to a single generator family.

### 4.4 Reinforcement Learning with Process Rewards

We further evaluate whether process rewards can provide effective fine-grained feedback for policy optimization. Following the verifier-gated strategy of VeriGate (Agrawal et al., 2026), we use PRM feedback only when all sampled responses for a prompt receive zero verifier reward, while retaining standard verifier-based GRPO updates otherwise. Unlike VeriGate, we directly use the PRM score at each reasoning step as the step-level reward without aggregating rewards from subsequent steps. We train on 8K prompts from DAPO-MATH-17K (Yu et al., 2026) and evaluate the resulting policy using zero-shot Avg@16 accuracy on six mathematical reasoning benchmarks.

**Analysis** Table 3 shows that RSP achieves the strongest overall performance, improving the average accuracy by 2.1% over PRM-guided with Qwen2.5-Math-PRM. The gain is particularly pronounced on AIME, where RSP outperforms Qwen2.5-Math-PRM by 7.8%. Supervised PRM reduces the average accuracy below that of the base policy, while OVM, Pseudo-Label PRM, and CRM$^\dagger$ all remain below GRPO on average. RSP is the only PRM-guided method that substantially improves the average accuracy over GRPO, indicating that introducing process rewards alone does not necessarily benefit policy optimization and that the quality of the process signal is critical. These results highlight the benefit of process rewards from RSP for policy optimization.

Table 3: Avg@16 accuracy (%) after reinforcement learning. All methods use the same policy initialization, training data, and optimization budget.

| Method | AIME | AMC | GSM8K | MATH500 | Minerva | Olympiad | Avg. |
|---|---|---|---|---|---|---|---|
| Base Policy | 23.3 | 70.0 | 92.5 | 78.6 | 38.2 | 52.4 | 59.2 |
| GRPO | 23.3 | 72.3 | **94.2** | <u>88.2</u> | **40.2** | 59.6 | 63.0 |
| Supervised PRM | 16.7 | 68.3 | 92.3 | 83.4 | 38.2 | 54.8 | 59.0 |
| OVM | 31.1 | 71.3 | 92.6 | 86.2 | 37.6 | 58.4 | 62.9 |
| Qwen2.5-Math-PRM | 31.1 | 68.3 | <u>93.8</u> | 86.8 | <u>39.8</u> | 59.8 | <u>63.3</u> |
| Pseudo-Label PRM | 26.7 | <u>72.3</u> | 92.6 | 85.8 | 35.8 | 57.3 | 61.8 |
| Joint-Supervised PRM | <u>31.1</u> | 70.0 | 92.3 | 87.2 | 38.2 | <u>59.8</u> | 63.1 |
| CRM$^{\dagger}$ | 23.3 | 70.0 | 92.5 | 86.2 | 37.0 | 57.3 | 61.1 |
| **RSP** | **38.9** | **72.5** | 92.9 | **88.4** | 39.3 | **60.2** | **65.4** |

### 4.5 Analysis of Reasoning State Propagation

**Transition Predictions** We examine whether the predicted transitions reflect meaningful changes in reasoning states. In particular, we compare the score distributions under the four state update patterns $\mathrm{G} \to \mathrm{G}$, $\mathrm{G} \to \mathrm{B}$, $\mathrm{B} \to \mathrm{B}$, and $\mathrm{B} \to \mathrm{G}$. Publicly available process-annotated data contain relatively few examples after reasoning has entered an invalid state, making $\mathrm{B} \to \mathrm{B}$ and $\mathrm{B} \to \mathrm{G}$ cases particularly scarce. We therefore use Qwen3-32B as an LLM judge to determine the validity of each reasoning step and construct approximately 10K step pairs covering the four state update patterns. As shown in Figure 2, for *break* predictions, $\mathrm{G} \to \mathrm{G}$ cases are concentrated at low scores, whereas $\mathrm{G} \to \mathrm{B}$ cases receive substantially higher scores. Similarly, *repair* scores are generally higher for $\mathrm{B} \to \mathrm{G}$ than for $\mathrm{B} \to \mathrm{B}$. The overall separation indicates that the learned transition scores capture meaningful changes in reasoning states.

**Scaling with Outcome Supervision** We further study how RSP benefits from increasing the amount of outcome supervision while keeping process supervision fixed. We vary the process-to-outcome data ratio from 1:1 to 1:9 and report the performance gain over the corresponding Process-Only model. Figure 2(c) shows that adding outcome supervision provides clear gains over Process-Only for both beam search and reinforcement learning. The gains generally increase as more outcome-annotated samples are introduced, showing that RSP can effectively benefit from additional outcome supervision for reasoning search and policy optimization.

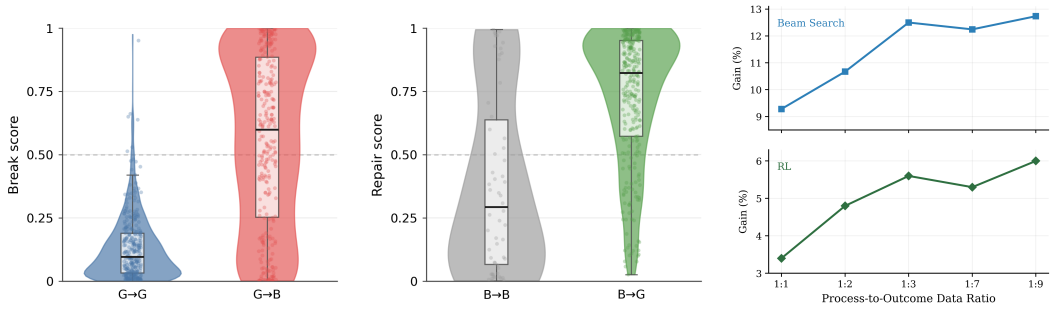
### 4.6 Ablation Study

Table 4 examines the contributions of the two supervision and the main design choices of RSP. Except for the results reported in Table 1, all subsequent beam search results use the Qwen3-4B PRM backbone with $b = 4$ and $e = 8$, and report the average accuracy on MATH500 and Gaokao.

**Supervision Sources** *Process-Only* and *Outcome-Only* use only process and outcome supervision, respectively. Process-Only shows large drops on beam search and ProcessBench (Zheng et al., 2024), whereas Outcome-Only nearly preserves Best-of-$N$ performance but drops by 9.6% on ProcessBench. This contrast suggests that outcome supervision provides strong signals for response selection and search, while process annotations remain important for fine-grained process evaluation. Combining the two sources of supervision gives the best performance in the evaluated settings.

**Repair Modeling** *No Repair* sets the repair probability to zero during training and inference, so an invalid reasoning state cannot return to a valid state. Removing repair degrades performance across all four settings, with the largest drop of 3.3% on beam search. This result shows that explicitly modeling error recovery is important for reasoning state propagation.

**Adjacent Representations** *Current Representation Only* predicts the break and repair probabilities using only the representation at the current reasoning boundary, rather than the representations




(a) Break prediction (b) Repair prediction (c) Outcome supervision scaling

Figure 2: (a) Distributions of predicted *break* scores for $\mathrm{G} \to \mathrm{G}$ and $\mathrm{G} \to \mathrm{B}$. (b) Distributions of predicted *repair* scores for $\mathrm{B} \to \mathrm{B}$ and $\mathrm{B} \to \mathrm{G}$. (c) Performance gains over Process-Only under increasing amounts of outcome supervision for beam search and reinforcement learning.

Table 4: Ablation study across evaluation settings. Colored numbers indicate changes relative to **Ours**, with darker colors denoting larger degradation.

| Variant | BoN | Beam Search | ProcessBench | RL |
|---|---|---|---|---|
| Process-Only | 72.0 (-2.4) | 59.7 (-12.5) | 58.3 (-7.5) | 59.8 (-5.6) |
| Outcome-Only | 74.2 (-0.2) | 69.5 (-2.7) | 56.2 (-9.6) | 61.6 (-3.8) |
| No Repair | 73.6 (-0.8) | 68.9 (-3.3) | 64.5 (-1.3) | 63.2 (-2.2) |
| Current Representation Only | 73.5 (-0.9) | 70.1 (-2.1) | 64.2 (-1.6) | 62.3 (-3.1) |
| Shared Token | 73.5 (-0.9) | 69.7 (-2.5) | 63.5 (-2.3) | 63.8 (-1.6) |
| No Outcome Propagation | 73.9 (-0.5) | 65.5 (-6.7) | 61.8 (-4.0) | 64.2 (-1.2) |
| **RSP** | **74.4** | **72.2** | **65.8** | **65.4** |

before and after the reasoning step. The consistent degradation shows that using both boundary representations provides more informative signal for estimating how the step updates reasoning states.

**Separate Break and Repair Tokens** *Shared Token* replaces the separate $\langle\mathrm{BREAK}\rangle$ and $\langle\mathrm{REPAIR}\rangle$ tokens with a single shared token. The resulting performance drops across all four settings, particularly 2.5% drop on beam search and 2.3% on ProcessBench, indicating that separate token representations help the model distinguish the roles of break and repair, while sharing a single token may introduce semantic interference between two conceptually different state transitions.

**Outcome Propagation** We further examine whether outcome supervision benefits intermediate state learning through the propagation mechanism. Specifically, we preserve the same forward state propagation while blocking backpropagation from the terminal outcome loss to earlier state transitions. This variant consistently degrades performance, with particularly large drops of 6.7% on beam search and 4.0% on ProcessBench. Since the forward propagation remains unchanged, the results show that propagating outcome supervision to earlier state updates is important for learning effective process rewards.

## 5 Conclusion

We introduced RSP, a reasoning state propagation approach for process reward modeling. By modeling *break* and *repair* probabilities, RSP explicitly links reasoning states across successive steps and captures error introduction, persistence, and recovery. The propagated reasoning state also provides a natural way to combine scarce process supervision with scalable outcome supervision, allowing terminal outcome supervision to provide learning signals to earlier state updates. Our results demonstrate the effectiveness of this formulation and highlight the complementary roles of process and outcome supervision in learning reliable process rewards.

### AI USE STATEMENT

We used generative AI tools primarily to assist with language editing and improving the clarity and presentation of the manuscript. Generative AI was also used in a limited experimental analysis, where Qwen3-32B served as an LLM judge as described in Section 4.5. All AI-assisted content and outputs were reviewed and verified by the authors, who take full responsibility for the final content of the paper.

### REPRODUCIBILITY STATEMENT

We provide the main implementation and experimental details in Section 4 and Appendix A, including training data, hyperparameters, evaluation settings, and baseline implementations. Code is provided in the supplementary material.

## REFERENCES

Aakriti Agrawal, Minghui Liu, and Furong Huang. Verigate: Verifier-gated step-level supervision for grpo. *arXiv preprint arXiv:2605.30451*, 2026.

Jie Cheng, Gang Xiong, Ruixi Qiao, Lijun Li, Chao Guo, Junle Wang, Yisheng Lv, and Fei-Yue Wang. Stop summation: Min-form credit assignment is all process reward model needs for reasoning. *Advances in Neural Information Processing Systems*, 38:131646–131671, 2026.

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

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Jinu Lee, Shivam Agarwal, Amruta Parulekar, Siddarth Madala, Dilek Hakkani-Tur, and Julia Hockenmaier. Reasoningflow: Discourse structures for understanding llm reasoning traces. *arXiv preprint arXiv:2606.05402*, 2026.

Weijie Li, Jin Wang, Liang-Chih Yu, and Xuejie Zhang. Step-grpo: Enhancing reasoning quality and efficiency via structured prm-based reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 31734–31742, 2026.

Wendi Li and Yixuan Li. Process reward model with q-value rankings. In *International Conference on Learning Representations*, volume 2025, pp. 14708–14726, 2025.

Minpeng Liao, Chengxi Li, Wei Luo, Wu Jing, and Kai Fan. Mario: Math reasoning with code interpreter output-a reproducible pipeline. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 905–924, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

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

Zihan Liu, Yang Chen, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acemath: Advancing frontier math reasoning with post-training and reward modeling, 2025b.

Jianqiao Lu, Zhiyang Dou, Hongru Wang, Zeyu Cao, Jianbo Dai, Yingjia Wan, Yunlong Feng, and Zhijiang Guo. Autopsv: Automated process-supervised verifier. *Advances in Neural Information Processing Systems*, 37:79935–79962, 2024.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

ByteDance Seed, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, et al. Seed1. 5-thinking: Advancing superb reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.13914*, 2025.

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

Shuaijie She, Junxiao Liu, Yifeng Liu, Jiajun Chen, Xin Huang, and Shujian Huang. R-prm: Reasoning-driven process reward modeling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13438–13451, 2025.

Charlie 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*, volume 2025, pp. 10131–10165, 2025.

Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *Advances in neural information processing systems*, 33: 596–608, 2020.

Xingwei Tan, Marco Valentino, Mahmud Elahi Akhter, Maria Liakata, and Nikolaos Aletras. Enhancing logical reasoning in language models via symbolically-guided monte carlo process supervision. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 31874–31888, 2025.

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

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

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

LLM Xiaomi, Bingquan Xia, Bowen Shen, Dawei Zhu, Di Zhang, Gang Wang, Hailin Zhang, Huaqiu Liu, Jiebao Xiao, Jinhao Dong, et al. Mimo: Unlocking the reasoning potential of language model–from pretraining to posttraining. *arXiv preprint arXiv:2505.07608*, 2025.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-evaluation guided beam search for reasoning. *Advances in Neural Information Processing Systems*, 36:41618–41650, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Xiaodan Liang, Yiwei Wang, and Jing Tang. Treerpo: Tree relative policy optimization. *arXiv preprint arXiv:2506.05183*, 2025b.

Zhangyue Yin, Qiushi Sun, Zhiyuan Zeng, Qinyuan Cheng, Xipeng Qiu, and Xuan-Jing Huang. Dynamic and generalizable process reward modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4203–4233, 2025.

Fei Yu, Anningzhe Gao, and Benyou Wang. Ovm, outcome-supervised value models for planning in mathematical reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 858–875, 2024.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024.

Zheng Zhang, Ziwei Shan, Kaitao Song, Yexin Li, and Kan Ren. Linking process to outcome: Conditional reward modeling for llm reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 87995–88016, 2025a.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516, 2025b.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*, 2024.

Jiachen Zhu, Congmin Zheng, Jianghao Lin, Kounianhua Du, Ying Wen, Yong Yu, Jun Wang, and Weinan Zhang. Retrieval-augmented process reward model for generalizable mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 8453–8468, 2025.

# A Additional Experimental Details

## A.1 RSP Training Details

Unless otherwise specified, we initialize RSP from Qwen3-4B and jointly optimize the backbone, the embeddings of the two special tokens, and the break and repair MLP heads. Following Section 3, each head receives the concatenation of the boundary representations before and after a reasoning step. We randomly select 20% of PRM800K and mix its process-annotated trajectories with outcome-annotated trajectories from AceMath-RM at a ratio of 1:3. Training proceeds for two epochs over the selected PRM800K subset using AdamW with a learning rate of $5 \times 10^{-6}$. We optimize only the process and outcome objectives in Section 3. Training is conducted in bfloat16 across eight NVIDIA H100 GPUs. We use the final checkpoint for evaluation. The code is provided in supplementary material.

**Transition Heads.** The break and repair predictors use two independent MLP heads with the same architecture. Given backbone hidden size $d$, each head takes the concatenated adjacent boundary representations in $\mathbb{R}^{2d}$ and applies a two-layer MLP, $2d \to d \to 1$, with GELU activation and dropout of $0.1$ after the input and hidden layer. The resulting scalar logits are converted to break and repair probabilities using a sigmoid.

## A.2 Reinforcement Learning Details

We initialize the policy from Qwen3-4B and perform full parameter training for one epoch on the 8K subset of DAPO-MATH-17K. For each prompt, we sample eight responses with a temperature of $1.0$ and top-$p$ of $1.0$. We use a rollout batch size of 16 and a training batch size of 128, with corresponding micro-batch sizes of 4 and 2. The maximum prompt and generation lengths are 512 and 4096 tokens, respectively. The actor learning rate is $1 \times 10^{-6}$ with a cosine schedule, and training is performed in bfloat16. During training, if all responses receive zero reward, we use RSP to score reasoning steps in each response. The step scores are normalized within the sampled

group and assigned to the tokens of their corresponding steps, without accumulating scores from subsequent steps. Otherwise, the update uses only the outcome rewards. We use the Dr. GRPO (Liu et al., 2025a) advantage estimator with a zero KL coefficient. All RL methods use the same policy training settings.

### A.3 Implementation Details of Additional Baselines

For a fair comparison, Pseudo-Label PRM, Joint-Supervised PRM, and CRM$^\dagger$ use the same backbone, training data mixture, and optimization settings as described above. Each baseline uses a single scalar step head evaluated at a `[PRM]` token appended after each reasoning step.

**Pseudo-Label PRM** For process-annotated trajectories, we apply BCE loss to all steps with available binary process labels. For outcome-annotated trajectories, the intermediate steps are treated as unlabeled, and hard pseudo labels are generated online from the current step predictions. A step is assigned a positive pseudo label when its predicted probability is at least $0.9$ and a negative pseudo label when the probability is at most $0.1$. The predictions between the two thresholds are ignored. Pseudo-labeling is enabled from the beginning of training.

**Joint-Supervised PRM** We use the same BCE loss as Supervised PRM on process-annotated trajectories. For each outcome-annotated trajectory, we additionally align the prediction at its final `[PRM]` position with the binary outcome label using BCE loss. Predictions at earlier steps of these trajectories are not supervised, and no pseudo labels are generated.

**CRM$^\dagger$** Following CRM (Zhang et al., 2025a), the scalar prediction at step $t$ parameterizes the conditional probability $h_t$ that the trajectory first enters an invalid state at that step. The probability of remaining valid through step $T$ is

$$S_T = \prod_{t=1}^{T} (1 - h_t). \tag{10}$$

For a process-annotated trajectory, we set its trajectory label to positive when all available step labels are valid and to negative otherwise. For a negative trajectory, the first step labeled invalid is used as the observed first-error position $z$, and CRM optimizes both the trajectory outcome likelihood and the localization likelihood $\Pr(z) = h_z \prod_{t<z}(1-h_t)$. CRM$^\dagger$ further applies the outcome objective to outcome-annotated trajectories without process labels: it minimizes $-\log S_T$ for a positive outcome and $-\log(1 - S_T)$ for a negative outcome, thereby marginalizing over the unknown first-error position.

### A.4 Outcome Data Construction

We convert the original AceMath-RM (Liu et al., 2025b) reward model training data into the outcome-annotated trajectories used in our experiments. For each example, we extract the user question and the corresponding assistant response, and retain the original binary correctness label as the trajectory-level outcome. We remove multiple-choice examples, malformed conversations, and responses for which reliable reasoning steps or a final answer cannot be identified.

**Outcome label reliability.** The outcome labels in AceMath-RM are constructed with multiple stages of quality control. Candidate responses are first labeled as correct or incorrect by comparing their answers against reference labels using the Qwen-Math evaluation toolkit. AceMath-RM further ranks positive and negative candidates using Qwen2.5-Math-RM-72B and applies score-sorted sampling to retain high-confidence responses, reducing label noise introduced by heuristic answer matching (Liu et al., 2025b). This ranking and filtering also helps reduce cases where a response reaches the correct final answer despite containing unresolved reasoning errors. We do not assume that a correct final answer implies that all preceding reasoning steps are valid. Instead, the outcome label is used only to supervise the final propagated state, while intermediate states are determined through state propagation and process supervision. Therefore, the filtered outcome labels provide reliable supervision for the final state in most cases.

Table 5: F1 scores (%) on ProcessBench. Our method classifies a step as valid when $p_t^{\mathrm{G}} \geq 0.5$. The last column reports the macro-average across all subsets.

| Method | GSM8K | MATH | OlympiadBench | Omni-MATH | Avg. |
|---|---|---|---|---|---|
| Supervised PRM | 69.8 | 60.3 | 48.5 | 47.9 | 56.6 |
| OVM | 49.9 | 44.1 | 28.1 | 28.2 | 37.6 |
| Qwen2.5-Math-PRM | **81.8** | 67.3 | **66.8** | **66.2** | **70.5** |
| Pseudo-Label PRM | 1.9 | 2.1 | 2.0 | 2.2 | 2.1 |
| Joint-Supervised PRM | 70.1 | 66.2 | 57.0 | 52.8 | 61.5 |
| CRM† | <u>73.8</u> | 56.2 | 30.7 | 24.4 | 46.3 |
| **RSP** | 73.5 | **71.5** | <u>60.9</u> | <u>57.1</u> | <u>65.8</u> |

## B Process Error Identification

We evaluate process error identification on ProcessBench (Zheng et al., 2024). ProcessBench requires a model to identify the first erroneous reasoning step or determine that the entire solution is correct. For RSP, we consider a reasoning prefix valid when $p_t^{\mathrm{G}} \geq 0.5$ and predict the first step with $p_t^{\mathrm{G}} < 0.5$ as the error location. If no reasoning step is identified as erroneous, the solution is predicted to be fully correct. Following ProcessBench, we report F1 as the harmonic mean of the accuracy on erroneous solutions and the accuracy on fully correct solutions.

**Analysis** Table 5 shows that, among the models trained under our shared experimental setup, RSP improves Supervised PRM by 9.2%. RSP also outperforms Qwen2.5-Math-PRM on MATH by 4.2%, although Qwen2.5-Math-PRM achieves the strongest overall result with additional large-scale process supervision. These results indicate that process annotations remain particularly important for ProcessBench, which explicitly evaluates fine-grained process error identification. The very low F1 of Pseudo-Label PRM further shows the difficulty of obtaining reliable fine-grained process supervision from pseudo labels.

## C Analysis of Process and Outcome Supervision

We analyze how process annotations and outcome labels jointly supervise break and repair in RSP. We first relate the fit of adjacent annotated states to the corresponding head predictions. We then derive how a final outcome changes the relative support for intermediate states and how this conditioning enters the training gradients.

We retain the notation of Section 3 and omit the trajectory index $i$. For a fixed question and reasoning trajectory $(x, \mathbf{s})$, state propagation gives

$$p_t^{\mathrm{G}} = p_{t-1}^{\mathrm{G}}(1-\alpha_t) + p_{t-1}^{\mathrm{B}}\beta_t, \qquad p_t^{\mathrm{B}} = 1 - p_t^{\mathrm{G}}. \tag{11}$$

All probabilities below refer to reasoning states along this observed trajectory.

### C.1 Process Supervision of Break and Repair

For an annotated step $t \in \mathcal{A}$, the individual state loss is

$$\ell_t^{\mathrm{step}} = -y_t^{\mathrm{step}} \log p_t^{\mathrm{G}} - (1 - y_t^{\mathrm{step}}) \log p_t^{\mathrm{B}}. \tag{12}$$

When $p_{t-1}^{\mathrm{G}} = 1$, propagation reduces to $p_t^{\mathrm{G}} = 1 - \alpha_t$, so this loss is binary cross-entropy for $\alpha_t$ with target $1 - y_t^{\mathrm{step}}$. When $p_{t-1}^{\mathrm{B}} = 1$, it instead reduces to binary cross-entropy for $\beta_t$ with target $y_t^{\mathrm{step}}$. Thus, fitting the current state supervises break after a valid prefix and repair after an invalid prefix. The same relation holds approximately when the preceding state is not deterministic.

**Proposition 1 (Adjacent-state supervision).** Suppose $t-1, t \in \mathcal{A}$, and let $a, b \in \mathcal{Z}$ denote the states specified by their process labels, with 1 mapped to G and 0 to B. If

$$p_{t-1}^{a} \geq 1 - \epsilon, \qquad p_t^{b} \geq 1 - \delta, \qquad 0 \leq \epsilon, \delta < 1, \tag{13}$$

then

$$\begin{aligned} y_{t-1}^{\text{step}} = 1 \quad &\Longrightarrow \quad \left|\alpha_t - (1 - y_t^{\text{step}})\right| \le \frac{\delta}{1-\epsilon}, \\ y_{t-1}^{\text{step}} = 0 \quad &\Longrightarrow \quad \left|\beta_t - y_t^{\text{step}}\right| \le \frac{\delta}{1-\epsilon}. \end{aligned} \tag{14}$$

**Proof.** Let $\bar{b}$ be the state other than $b$. Since all entries of $\mathbf{M}_t$ are nonnegative,

$$\delta \ge p_t^{\bar{b}} \ge p_{t-1}^{a} \mathbf{M}_t[a, \bar{b}] \ge (1-\epsilon)\big(1 - \mathbf{M}_t[a, b]\big). \tag{15}$$

Hence $1 - \mathbf{M}_t[a, b] \le \delta/(1-\epsilon)$. Substituting the appropriate entry of $\mathbf{M}_t$ gives both inequalities in equation 14. For example, when an invalid prefix is followed by a valid one and both states are well fitted, the bound requires $\beta_t$ to be close to one. The result applies to the prediction conditioned on the annotated preceding state. It establishes how process supervision constrains the meaning of break and repair through state propagation, without requiring separate labels for the two heads.

### C.2 Outcome Evidence for Intermediate Reasoning States

For an outcome-annotated trajectory, let $c = \mathrm{G}$ when $y^{\text{out}} = 1$ and $c = \mathrm{B}$ otherwise, following the terminal target used by $\mathcal{L}_{\text{out}}$. Expanding the propagation product gives

$$Z_{\text{out}} \coloneqq p_T^c = \sum_{z_{1:T-1} \in \mathcal{Z}^{T-1}} \prod_{j=1}^{T} \mathbf{M}_j[z_{j-1}, z_j], \tag{16}$$

where $z_0 = \mathrm{G}$ and $z_T = c$ are fixed. The individual outcome loss is $\ell^{\text{out}} = -\log Z_{\text{out}}$. The sum includes every possible sequence of intermediate states ending in $c$. A correct outcome can therefore be explained by reasoning that remains valid throughout or by reasoning that contains an error followed by recovery.

To compare these explanations at position $t$, define

$$h_t(a) \coloneqq \big(\mathbf{M}_{t+1} \cdots \mathbf{M}_T\big)[a, c], \qquad a \in \mathcal{Z}, \tag{17}$$

with the empty product equal to the identity matrix. The quantity $h_t(a)$ is the probability of reaching the observed terminal state from state $a$ through the remaining reasoning steps. It satisfies

$$h_T(a) = \mathbf{1}\{a = c\}, \qquad h_{t-1}(a) = \sum_{b \in \mathcal{Z}} \mathbf{M}_t[a, b] h_t(b). \tag{18}$$

The joint probability of $z_t = a$ and $z_T = c$ is $p_t^a h_t(a)$. Conditioning on the terminal label therefore gives

$$\gamma_t^a \coloneqq p_\theta(z_t = a \mid z_T = c, x, \mathbf{s}) = \frac{p_t^a h_t(a)}{Z_{\text{out}}}, \qquad Z_{\text{out}} = \sum_{a \in \mathcal{Z}} p_t^a h_t(a). \tag{19}$$

This expression combines the model's prefix evaluation $p_t^a$ with the likelihood of the final result under that state, $h_t(a)$. In particular,

$$\gamma_t^{\mathrm{G}} - p_t^{\mathrm{G}} = \frac{p_t^{\mathrm{G}} p_t^{\mathrm{B}} \big(h_t(\mathrm{G}) - h_t(\mathrm{B})\big)}{Z_{\text{out}}}. \tag{20}$$

The outcome **increases the relative support for whichever intermediate state makes the observed final result more likely under the remaining reasoning**. It leaves the state probabilities unchanged when the two explanations have equal likelihood. Thus, the terminal observation can support or revise a prefix prediction rather than simply reinforce its existing preference. These posterior probabilities describe conditioning at fixed model parameters, and the process score remains $p_t^{\mathrm{G}}$, computed from the prefix alone.

### C.3 Learning Break and Repair from the Final Outcome

The posterior reweighting above also determines the gradients of the outcome loss. Let $\eta_t^{\mathrm{br}}$ and $\eta_t^{\mathrm{rp}}$ be the head logits, with $\alpha_t = \sigma(\eta_t^{\mathrm{br}})$ and $\beta_t = \sigma(\eta_t^{\mathrm{rp}})$. The outcome-conditioned targets for break and repair are

$$\widetilde{\alpha}_t = \frac{\alpha_t h_t(\mathrm{B})}{h_{t-1}(\mathrm{G})}, \qquad \widetilde{\beta}_t = \frac{\beta_t h_t(\mathrm{G})}{h_{t-1}(\mathrm{B})}. \tag{21}$$

For finite logits, both denominators are positive. Where the corresponding preceding state has positive probability, these quantities are respectively $p_\theta(z_t = \mathrm{B} \mid z_{t-1} = \mathrm{G}, z_T = c, x, \mathbf{s})$ and $p_\theta(z_t = \mathrm{G} \mid z_{t-1} = \mathrm{B}, z_T = c, x, \mathbf{s})$.

**Proposition 2 (Outcome supervision of the two heads).** The local derivatives of the outcome loss are

$$\begin{aligned} \frac{\partial \ell^{\mathrm{out}}}{\partial \eta_t^{\mathrm{br}}} &= \gamma_{t-1}^{\mathrm{G}}\big(\alpha_t - \widetilde{\alpha}_t\big), \\ \frac{\partial \ell^{\mathrm{out}}}{\partial \eta_t^{\mathrm{rp}}} &= \gamma_{t-1}^{\mathrm{B}}\big(\beta_t - \widetilde{\beta}_t\big). \end{aligned} \tag{22}$$

**Proof.** Holding the other head outputs fixed, write

$$\begin{aligned} Z_{\mathrm{out}} = p_{t-1}^{\mathrm{G}}\big((1-\alpha_t)h_t(\mathrm{G}) + \alpha_t h_t(\mathrm{B})\big) \\ + p_{t-1}^{\mathrm{B}}\big(\beta_t h_t(\mathrm{G}) + (1-\beta_t)h_t(\mathrm{B})\big). \end{aligned} \tag{23}$$

Differentiating $-\log Z_{\mathrm{out}}$ through the sigmoid gives

$$\begin{aligned} \frac{\partial \ell^{\mathrm{out}}}{\partial \eta_t^{\mathrm{br}}} &= \frac{p_{t-1}^{\mathrm{G}}\alpha_t(1-\alpha_t)}{Z_{\mathrm{out}}}\big(h_t(\mathrm{G}) - h_t(\mathrm{B})\big), \\ \frac{\partial \ell^{\mathrm{out}}}{\partial \eta_t^{\mathrm{rp}}} &= -\frac{p_{t-1}^{\mathrm{B}}\beta_t(1-\beta_t)}{Z_{\mathrm{out}}}\big(h_t(\mathrm{G}) - h_t(\mathrm{B})\big). \end{aligned} \tag{24}$$

Substituting equation 18, equation 19, and equation 21 yields equation 22.

Equation 22 has the form of a weighted binary cross-entropy gradient. Each head is compared with its outcome-conditioned target and weighted by the posterior probability of the relevant preceding state. For example, a repair target larger than $\beta_t$ gives a negative derivative for the repair logit, encouraging a higher repair prediction. If $h_t(\mathrm{G}) = h_t(\mathrm{B}) > 0$, the two targets equal the current predictions and both local derivatives vanish. The final result therefore supplies a learning signal according to how a step relates to the subsequent reasoning, rather than imposing the same validity target at every position.

These equalities describe the gradient of the existing outcome objective. In the weighted cross-entropy interpretation, the posterior quantities are evaluated at the current parameters and treated as fixed targets and weights. Automatic differentiation of $\ell^{\mathrm{out}}$ produces the same local derivatives without an explicit posterior-labeling stage.

**Implication for joint supervision.** Process annotations provide direct evidence about intermediate validity through equation 14. On outcome-only data, the same model supplies the prefix probabilities used in equation 19, while the terminal label reweights these predictions through the remaining reasoning. The two losses consequently train the same break and repair heads using different levels of annotation.

A useful instance is a correct outcome following a predicted invalid prefix. If $y^{\mathrm{out}} = 1$, $p_k^{\mathrm{B}} \geq 1 - \epsilon$, and $p_T^{\mathrm{G}} \geq 1 - \delta$, with $0 \leq \epsilon, \delta < 1$, then

$$h_k(\mathrm{B}) \geq 1 - \frac{\delta}{1-\epsilon}. \tag{25}$$

This follows from $p_T^{\mathrm{B}} \geq p_k^{\mathrm{B}}(1 - h_k(\mathrm{B}))$. Conditional on retaining the invalid prefix prediction, fitting the correct outcome requires the remaining steps to provide sufficient recovery. The outcome constrains their combined effect while leaving the location of recovery to be learned from the reasoning

context. This connects locally supervised validity judgments with terminal feedback on trajectories that lack process annotations. Section 4.5 and Table 4 evaluate the learned break and repair scores and the empirical contribution of the two supervision sources.

### C.4 Strength of State Propagation

The analysis above shows that outcome supervision reaches intermediate transitions through the propagated reasoning states. We further examine the strength of this propagation in the learned model. From equation 11, using $p^{\mathrm{B}}_{t-1} = 1 - p^{\mathrm{G}}_{t-1}$, we can write

$$p^{\mathrm{G}}_t = \beta_t + \kappa_t p^{\mathrm{G}}_{t-1}, \qquad \kappa_t \coloneqq 1 - \alpha_t - \beta_t. \tag{26}$$

Therefore,

$$\frac{\partial p^{\mathrm{G}}_t}{\partial p^{\mathrm{G}}_{t-1}} = \kappa_t. \tag{27}$$

The coefficient $\kappa_t$ directly measures how strongly the propagated state after step $t$ depends on the preceding propagated state. Across multiple steps,

$$\frac{\partial p^{\mathrm{G}}_t}{\partial p^{\mathrm{G}}_s} = \prod_{j=s+1}^{t} \kappa_j, \qquad s < t. \tag{28}$$

Thus, smaller $|\kappa_t|$ weakens the influence of earlier reasoning states, whereas larger $|\kappa_t|$ preserves more of the preceding state information.

The same coefficient also determines how outcome supervision is transmitted backward through state propagation. Recall that $h_t(a)$ denotes the probability of reaching the observed terminal state from state $a$. From the backward recursion in equation 18, the difference between the two possible current states satisfies

$$h_{t-1}(\mathrm{G}) - h_{t-1}(\mathrm{B}) = \kappa_t\big(h_t(\mathrm{G}) - h_t(\mathrm{B})\big). \tag{29}$$

Consequently,

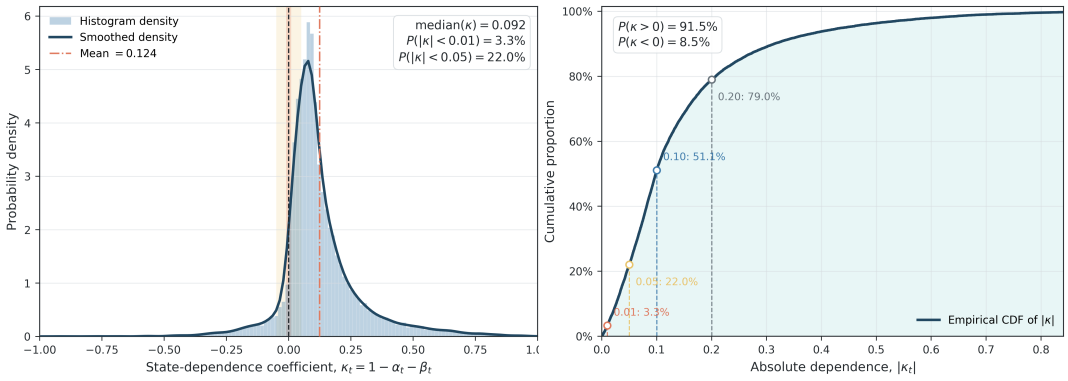
$$|h_t(\mathrm{G}) - h_t(\mathrm{B})| = \prod_{j=t+1}^{T} |\kappa_j|, \tag{30}$$

where the terminal difference has magnitude one. Since $h_t(\mathrm{G}) - h_t(\mathrm{B})$ appears directly in the outcome gradients in equation 24, $\kappa_t$ has a dual role: it controls both the forward dependence of later states on earlier states and the backward propagation of outcome supervision to earlier transitions. Importantly, this formulation does not require outcome supervision to propagate equally strongly to all preceding transitions. Its influence is modulated by the learned intervening state transitions, so that earlier states whose effect is weakly preserved receive correspondingly weaker terminal evidence.

**Empirical Analysis** Figure 3(a) shows that the learned state dependence is predominantly positive. The median value of $\kappa_t$ is $0.092$ and the mean is $0.124$. Moreover, $91.5\%$ of the transitions have $\kappa_t > 0$, while only $8.5\%$ have $\kappa_t < 0$. Therefore, for the large majority of reasoning steps, a higher probability of being in the valid state before the step continues to increase the propagated valid state probability after the step. The distribution also contains a broad positive tail, showing that a subset of transitions retains substantially stronger dependence on preceding states.

Figure 3(b) further shows that the propagation strength varies considerably across reasoning steps. Only $3.3\%$ of transitions satisfy $|\kappa_t| < 0.01$, indicating that transitions that almost completely remove dependence on the preceding propagated state are relatively uncommon. Meanwhile, $22.0\%$ of transitions have $|\kappa_t| < 0.05$, $51.1\%$ have $|\kappa_t| < 0.1$, and $79.0\%$ have $|\kappa_t| < 0.2$. Thus, RSP does not maintain the same propagation strength at every reasoning step. Instead, some transitions substantially reduce the influence of preceding states, while others preserve stronger dependence across successive steps.

These results suggest that reasoning state propagation is selective rather than uniform. This behavior is consistent with the formulation of RSP: later reasoning may preserve an earlier reasoning state, introduce an error, or recover from an earlier error, and these cases need not retain the same amount




(a) Distribution of $\kappa_t$ (b) Empirical cumulative distribution of $|\kappa_t|$

Figure 3: (a) Distribution of $\kappa_t = 1 - \alpha_t - \beta_t$. The signed value indicates the direction and strength of the dependence of the updated propagated state on the preceding state. (b) Empirical cumulative distribution of $|\kappa_t|$. Smaller $|\kappa_t|$ indicates weaker dependence of the updated propagated state on the preceding propagated state.

of dependence on the preceding state. The same learned transition structure also determines how outcome supervision is propagated backward. Therefore, RSP allows both state dependence and outcome supervision to vary across the reasoning trajectory according to the learned state transitions, rather than requiring equally strong propagation at every step.

## D Scaling Analysis

We further examine how RSP behaves under different model capacities and amounts of training supervision. Unless otherwise specified, all training and evaluation settings follow the main experiments. *Process-Only* denotes the corresponding model trained with process supervision alone.

**Scaling with Model Size.** We first study whether the benefit of RSP persists as the PRM backbone scales. We vary the backbone size of the Qwen3 series models from 1.7B to 8B while keeping the training data and other settings fixed. As shown in Figure 4, RSP consistently improves as the model size increases across all four evaluation settings. Compared with Process-Only, RSP maintains a clear advantage across all model sizes. These results indicate that the benefit of incorporating outcome supervision is maintained as the PRM capacity increases.

**Scaling with Outcome Supervision.** We further study how RSP scales with the amount of outcome supervision by varying the process-to-outcome data ratio in each training batch from 1:1 to 1:9, while keeping the process supervision fixed. As shown in Figure 5, increasing outcome supervision provides substantial improvements in beam search and reinforcement learning. Beam search accuracy increases from approximately 69% to above 72%, while reinforcement learning improves from about 63.2% to 65.8%. In contrast, Best-of-$N$ selection and ProcessBench remain relatively stable across different ratios. These results show that RSP can effectively benefit from additional outcome supervision, particularly for reasoning search and policy optimization.

## E Training Data and Evaluation Overlap

We further clarify the relation between the training data used in our experiments and the evaluation benchmarks.

**Process Supervision.** For process supervision, we use a deduplicated version of PRM800K (Lightman et al., 2024) before selecting the 20% subset used in our main experiments. Specifically, duplicated full reasoning trajectories are removed during preprocessing. The deduplication is performed at the trajectory level. Distinct trajectories that share partial reasoning prefixes are retained, since they correspond to different annotated reasoning trajectories. PRM800K

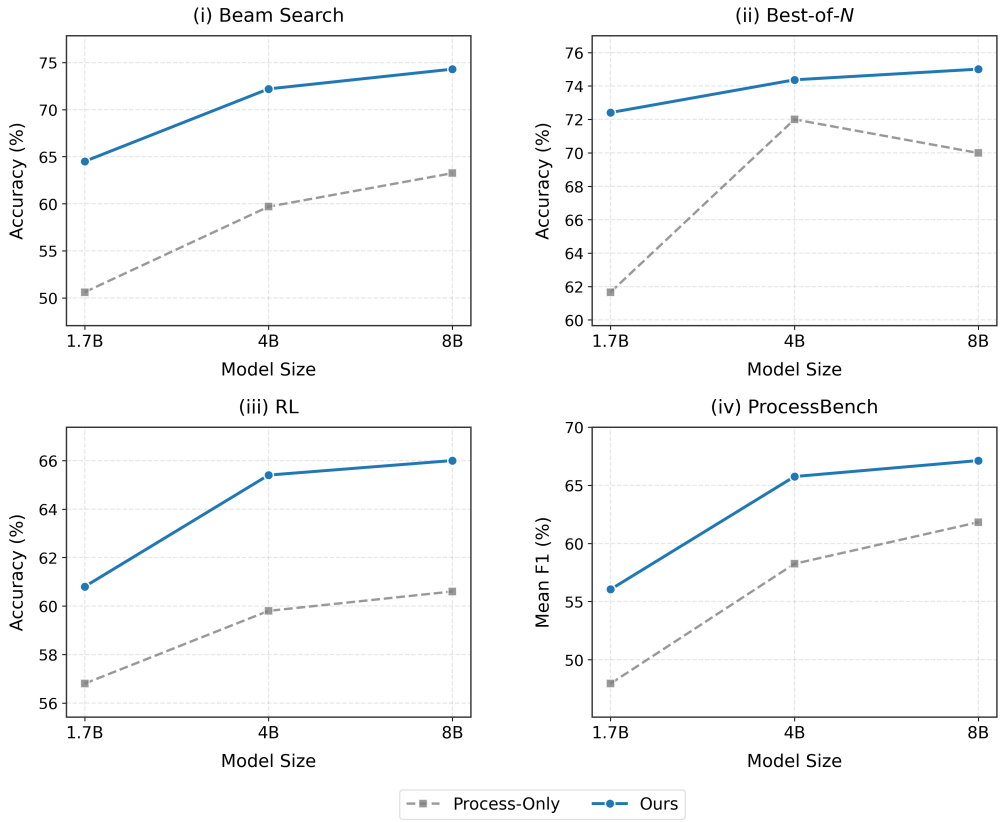



Figure 4: Performance with different PRM backbone sizes. We compare RSP with Process-Only on beam search, Best-of-$N$ selection, reinforcement learning, and ProcessBench.

uses the MATH split released by Lightman et al. (2024). In this split, 4,500 problems from the original MATH test set are included in the training set, while the remaining 500 problems are held out for evaluation. These held-out problems form MATH500 and are not included in the PRM800K training data used for process supervision. Therefore, our MATH500 evaluation is separated from the PRM800K process training data by construction.

**Outcome Supervision.** For outcome supervision, we use the released AceMath-RM (Liu et al., 2025b) training data and convert it into outcome-annotated trajectories following Appendix A.4. Our preprocessing only filters examples and converts each retained response into reasoning steps with a trajectory-level outcome label. It does not introduce additional questions from any evaluation benchmark. AceMath-RM is constructed from the math training data of AceMath, where test-data filtering is applied before model training.

**Reinforcement Learning.** The verifier-gated update used in our reinforcement learning experiments follows VeriGate (Agrawal et al., 2026). For RL training data, we use an 8K subset of the public DAPO-MATH-17K dataset (Yu et al., 2026), following the setting described in Section 4.4 and Appendix A.2. DAPO-MATH-17K is released as a curated and processed math training dataset. We sample the 8K training prompts only from this released training set. All methods in Table 3 use exactly the same 8K training prompts, together with the same policy initialization and optimization budget.

Overall, the process, outcome, and reinforcement learning experiments use their corresponding public training data. Our preprocessing does not add examples from the evaluation benchmarks to these training sets. In particular, MATH500 is explicitly held out from the PRM800K process training split. The same training data are used across the compared methods under each experimental setting.

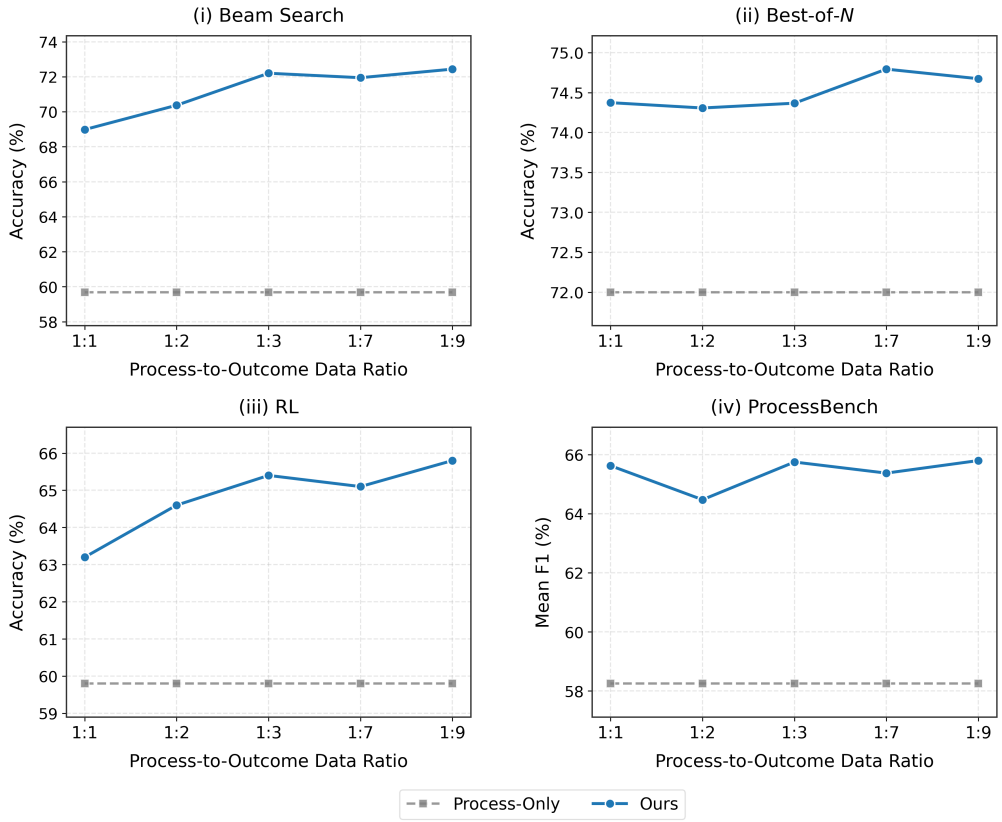



Figure 5: Performance with different amounts of outcome supervision. The horizontal axis denotes the process-to-outcome data ratio within each training batch. For example, 1:3 indicates that the numbers of process-annotated and outcome-annotated examples are mixed at a ratio of 1:3 in a batch. The process supervision is kept fixed while the amount of outcome supervision is varied.

## F Qualitative Examples

We provide qualitative examples to illustrate how RSP propagates reasoning validity across different reasoning patterns. Each step reports the propagated process score together with the predicted *break* and *repair* probabilities. Examples illustrate representative state patterns, including validity preservation, error introduction and persistence, and error recovery.

**Example 1**

**Question:**
How many four-digit positive integers contain at least one 2 or 3 as a digit?

**LLM Response:**

**Step 1:** ✓ **PRM** 0.992 **Break** 0.008 **Repair** 0.909
To determine how many four-digit positive integers contain at least one 2 or 3 as a digit, we can use the principle of complementary counting. This involves first finding the total number of four-digit positive integers and then subtracting the number of four-digit positive integers that do not contain any 2s or 3s as digits.

**Step 2:** ✓ **PRM** 0.997 **Break** 0.002 **Repair** 0.910
Calculate the total number of four-digit positive integers.

A four-digit positive integer ranges from 1000 to 9999. Therefore, the total number of four-digit positive integers is:

$$9999 - 1000 + 1 = 9000.$$

**Step 3:** ✓ **PRM** 0.995 **Break** 0.005 **Repair** 0.905
Calculate the number of four-digit positive integers that do not contain any 2s or 3s as digits.
For a four-digit number, the digits can be chosen as follows:

- The thousands place can be any digit from 1 to 9 except 2 and 3, so there are 7 choices $(1, 4, 5, 6, 7, 8, 9)$.
- The hundreds, tens, and units places can each be any digit from 0 to 9 except 2 and 3, so there are 8 choices for each of these places $(0, 1, 4, 5, 6, 7, 8, 9)$.

**Step 4:** ✓ **PRM** 0.996 **Break** 0.003 **Repair** 0.887
Thus, the total number of four-digit positive integers that do not contain any 2s or 3s as digits is:

$$7 \times 8 \times 8 \times 8 = 7 \times 512 = 3584.$$

**Step 5:** ✓ **PRM** 0.997 **Break** 0.002 **Repair** 0.853
Calculate the number of four-digit positive integers that contain at least one 2 or 3 as a digit.
Using the principle of complementary counting, we subtract the number of four-digit positive integers that do not contain any 2s or 3s from the total number of four-digit positive integers:

$$9000 - 3584 = 5416.$$

**Step 6:** ✓ **PRM** 1.000 **Break** 0.000 **Repair** 0.835
Therefore, the number of four-digit positive integers that contain at least one 2 or 3 as a digit is:

$$\boxed{5416}.$$

G → G**: Validity Preservation.** Example 1 shows a fully valid reasoning trajectory. The break probability remains close to zero throughout the solution, and the propagated process score stays near one, correctly preserving the valid reasoning state across successive steps.

**Example 2**

**Question:**
Given that the function $f(x)$ defined on $\mathbb{R}$ is an odd function and satisfies $f(1 + x) = f(3 + x)$. When $0 \leq x \leq 1$, $f(x) = x^3 - x$. Find

$$f\left(\frac{11}{2}\right) + f(6).$$

**LLM Response:**

**Step 1:** ✓ **PRM** 0.991 **Break** 0.009 **Repair** 0.973
To find $f\left(\frac{11}{2}\right) + f(6)$, we need to use the properties of the function $f$. The function $f$ is an odd function, which means $f(-x) = -f(x)$, and it satisfies the periodicity condition $f(1 + x) = f(3 + x)$. This implies that $f(x) = f(x + 2)$, so $f$ is periodic with period 2.

**Step 2:** ✓ **PRM** 0.992 **Break** 0.008 **Repair** 0.998
First, let's find $f(6)$. Since $f$ is periodic with period 2, we have:

$$f(6) = f(6 - 2 \cdot 3) = f(0).$$

Given that $f(x) = x^3 - x$ for $0 \le x \le 1$, we substitute $x = 0$:

$$f(0) = 0^3 - 0 = 0.$$

So, $f(6) = 0$.

**Step 3:** × **PRM** 0.104 **Break** 0.895 **Repair** 0.011

Next, let's find $f\left(\frac{11}{2}\right)$. Using the periodicity of $f$, we have:

$$f\left(\frac{11}{2}\right) = f\left(\frac{11}{2} - 2 \cdot 2\right) = f\left(\frac{11}{2} - 4\right) = f\left(\frac{3}{2}\right).$$

Since $f$ is odd, we can write:

$$f\left(\frac{3}{2}\right) = f\left(1 + \frac{1}{2}\right) = f\left(3 + \frac{1}{2} - 2\right) = f\left(1 + \frac{1}{2}\right) = f\left(\frac{1}{2}\right).$$

Given that $f(x) = x^3 - x$ for $0 \le x \le 1$, we substitute $x = \frac{1}{2}$:

$$f\left(\frac{1}{2}\right) = \left(\frac{1}{2}\right)^3 - \frac{1}{2} = \frac{1}{8} - \frac{1}{2} = \frac{1}{8} - \frac{4}{8} = -\frac{3}{8}.$$

So,

$$f\left(\frac{11}{2}\right) = -\frac{3}{8}.$$

**Step 4:** × **PRM** 0.018 **Break** 0.870 **Repair** 0.005

Finally, we add $f\left(\frac{11}{2}\right)$ and $f(6)$:

$$f\left(\frac{11}{2}\right) + f(6) = -\frac{3}{8} + 0 = -\frac{3}{8}.$$

Thus, the answer is:

$$\boxed{-\frac{3}{8}}.$$

**G → B and B → B: Error Introduction and Persistence.** In Example 2, the reasoning is valid through Step 2, but Step 3 incorrectly reduces $f(3/2)$ to $f(1/2)$. The high break probability sharply lowers the process score, while the subsequent step continues from the incorrect result with a very low repair probability, preserving the invalid reasoning state.

**Example 3**

**Question:**

Rationalize the denominator of

$$\frac{7}{\sqrt{75}}.$$

**LLM Response:**

**Step 1:** ✓ **PRM** 0.982 **Break** 0.018 **Repair** 0.762

To rationalize the denominator of the expression $\frac{7}{\sqrt{75}}$, we need to eliminate the square root from the denominator. We can do this by multiplying both the numerator and the denominator by $\sqrt{75}$.

Here are the steps:

Start with the original expression:

$$\frac{7}{\sqrt{75}}.$$

**Step 2:** ✓ **PRM** 0.995 **Break** 0.004 **Repair** 0.941

Multiply both the numerator and the denominator by $\sqrt{75}$:

$$\frac{7}{\sqrt{75}} \cdot \frac{\sqrt{75}}{\sqrt{75}}.$$

**Step 3:** ✓ **PRM** 0.987 **Break** 0.012 **Repair** 0.738

Perform the multiplication in the numerator and the denominator:

$$\frac{7 \cdot \sqrt{75}}{\sqrt{75} \cdot \sqrt{75}} = \frac{7\sqrt{75}}{75}.$$

**Step 4:** × **PRM** 0.399 **Break** 0.600 **Repair** 0.358

Simplify the expression by canceling the common factor of 7 in the numerator and the denominator:

$$\frac{7\sqrt{75}}{75}.$$

**Step 5:** ✓ **PRM** 0.802 **Break** 0.018 **Repair** 0.682

Simplify $\sqrt{75}$. We know that $75 = 25 \cdot 3$ and $\sqrt{25} = 5$, so:

$$\sqrt{75} = \sqrt{25 \cdot 3} = 5\sqrt{3}.$$

**Step 6:** ✓ **PRM** 0.924 **Break** 0.009 **Repair** 0.651

Substitute $\sqrt{75}$ with $5\sqrt{3}$ in the expression:

$$\frac{7 \cdot 5\sqrt{3}}{75} = \frac{35\sqrt{3}}{75}.$$

**Step 7:** ✓ **PRM** 0.925 **Break** 0.044 **Repair** 0.556

Simplify the fraction $\frac{35}{75}$. The greatest common divisor of 35 and 75 is 5, so:

$$\frac{35}{75} = \frac{35 \div 5}{75 \div 5} = \frac{7}{15}.$$

**Step 8:** ✓ **PRM** 0.972 **Break** 0.024 **Repair** 0.929

Therefore, the expression simplifies to:

$$\frac{7\sqrt{3}}{15}.$$

**Step 9:** ✓ **PRM** 0.999 **Break** 0.000 **Repair** 0.972

The rationalized form of the expression is

$$\boxed{\frac{7\sqrt{3}}{15}}.$$

B → G**: Error Recovery.** Example 3 illustrates recovery after an intermediate error. Step 4 introduces an invalid reasoning step and reduces the process score, whereas the subsequent valid derivation returns to a correct computation. Correspondingly, the repair probability increases and the propagated process score gradually recovers.

**Example 4**

**Question:**
Calculate the sum of all integral values of $c$ with $c \leq 20$ for which the equation

$$y = x^2 - 9x - c$$

produces two rational roots.

**LLM Response:**

**Step 1:** ✓ **PRM** 0.955 **Break** 0.045 **Repair** 0.890
To determine the sum of all integral values of $c$ with $c \leq 20$ for which the quadratic equation $y = x^2 - 9x - c$ has two rational roots, we start by using the discriminant. A quadratic equation $ax^2 + bx + c = 0$ has two rational roots if and only if its discriminant

$$\Delta = b^2 - 4ac$$

is a perfect square.

**Step 2:** ✓ **PRM** 0.985 **Break** 0.012 **Repair** 0.914
For the given equation $x^2 - 9x - c = 0$, the discriminant is

$$\Delta = (-9)^2 - 4 \cdot 1 \cdot (-c) = 81 + 4c.$$

We need $81 + 4c$ to be a perfect square. Let

$$81 + 4c = k^2$$

for some integer $k$. Then

$$c = \frac{k^2 - 81}{4}.$$

Since $c$ must be an integer and $81 \equiv 1 \pmod 4$, we require

$$k^2 \equiv 1 \pmod 4.$$

Therefore, $k$ must be an odd integer.

**Step 3:** × **PRM** 0.431 **Break** 0.570 **Repair** 0.500
Next, we need to find the values of $k$ such that $c \leq 20$. Substituting $c = \frac{k^2-81}{4}$ into the inequality gives

$$\frac{k^2 - 81}{4} \leq 20 \quad \Rightarrow \quad k^2 \leq 161.$$

The largest odd integer $k$ that satisfies $k^2 \leq 161$ is 12, and the smallest odd integer is $-13$. Therefore, $k$ can take any odd value from $-13$ to 13.

**Step 4:** ✓ **PRM** 0.890 **Break** 0.008 **Repair** 0.812
We now list the odd integers in this range and calculate the corresponding values of $c$:

$$\begin{aligned}
k = -13 \quad &\Rightarrow \quad c = \frac{169 - 81}{4} = 22 \qquad \text{(not valid since } c \leq 20\text{)}, \\
k = -11 \quad &\Rightarrow \quad c = \frac{121 - 81}{4} = 10, \\
k = -9 \quad &\Rightarrow \quad c = \frac{81 - 81}{4} = 0, \\
k = -7 \quad &\Rightarrow \quad c = \frac{49 - 81}{4} = -8, \\
k = -5 \quad &\Rightarrow \quad c = \frac{25 - 81}{4} = -14, \\
k = -3 \quad &\Rightarrow \quad c = \frac{9 - 81}{4} = -18, \\
k = -1 \quad &\Rightarrow \quad c = \frac{1 - 81}{4} = -20.
\end{aligned}$$

The positive values of $k$ give the same values of $c$. In particular, $k = 13$ also gives $c = 22$, which is not valid since $c \leq 20$. Therefore, the valid values of $c$ are

$$-20,\ -18,\ -14,\ -8,\ 0,\ 10.$$

**Step 5:** ✓ **PRM** 0.978 **Break** 0.000 **Repair** 0.805

Summing these values, we get

$$-20 + (-18) + (-14) + (-8) + 0 + 10 = -50.$$

Thus, the sum of all integral values of $c$ is

$$\boxed{-50}.$$

B $\rightarrow$ G**: Error Recovery.** Example 4 illustrates recovery after an intermediate error. Step 3 incorrectly includes $\pm 13$ among the possible values of $k$, even though $13^2 > 161$. In Step 4, the response evaluates these values, obtains $c = 22$, and explicitly excludes them because $c \leq 20$. The derivation then returns to the correct set of values and produces the correct final answer.