Unearthing Gems from Stones: Policy Optimization with Negative Sample Augmentation for LLM Reasoning

Anonymous ACL submission

Abstract

Recent advances in reasoning language models have witnessed a paradigm shift from short to long CoT pattern. Given the substantial computational cost of rollouts in long CoT models, maximizing the utility of fixed training datasets becomes crucial. Our analysis reveals that negative responses contain valuable components such as self-reflection and error-correction steps, yet primary existing methods either completely discard negative samples (RFT) or apply equal penalization across all tokens (RL), failing to leverage these potential learning signals. In light of this, we propose Behavior Constrained Policy Gradient with Negative Sample Augmentation (BCPG-NSA), a fine-grained offline RL framework that encompasses three stages: 1) sample segmentation, 2) consensus-based step correctness assessment combining LLM and PRM judgers, and 3) policy optimization with NSA designed to effectively mine positive steps within negative samples. Experimental results show that BCPG-NSA outperforms baselines on several challenging math/coding reasoning benchmarks using the same training dataset, achieving improved sample efficiency and demonstrating robustness and scalability when extended to multiple iterations.

1 Introduction

Reasoning capabilities are a critical aspect of evaluating the intelligence of large language models (LLMs). Recent advances have witnessed a paradigm shift from short to long chain-of-thought (CoT) reasoning, particularly after the release of OpenAI's o1 series (OpenAI, 2024) and Deepseek R1 (Guo et al., 2025). However, generating lengthy CoT responses incurs substantial computational costs. In the online learning paradigm, the periodic rollout process has become the bottleneck of post-training system (Luo et al., 2025), significantly reducing training efficiency. Consequently, maximizing the utility of fixed training datasets under offline learning paradigm becomes increasingly crucial.

Motivated by this, a natural question arises: *How can we better utilize negative samples, particularly those from long CoT model?* Currently, self-improvement algorithms for enhancing LLMs' reasoning capabilities fall into two categories: rejection sampling fine-tuning (RFT (Zhang et al.; Tian et al., 2025)) and reinforcement learning (RL (Rafailov et al., 2023; Roux et al., 2025; Team, 2025)), yet neither fully exploits the potential of negative samples. For RFT methods, negative samples are entirely discarded. While RL methods like DPO (Rafailov et al., 2023), GRPO (Shao et al., 2024), and GPG (Chu et al., 2025) attempt to leverage negative samples, they simply apply equal penalization to all tokens without fine-grained discrimination.

We argue that while treating all steps in negative samples as flawed is reasonable for short CoT responses due to their brevity and directness, the situation differs significantly in long reasoning trajectories. Even when the final answer is incorrect, many intermediate steps can be valuable (Li et al., 2025; Ahmad et al., 2025). As illustrated in Figure 1(a), we observe that model responses often exhibit intrinsic behaviors such as verification, error-correction, and self-reflection (Min et al., 2024; Ahmad et al., 2025), with these patterns occurring more frequently than in positive samples, partly due to the typically longer response length of negative samples (Fatemi et al., 2025). Therefore, in the long CoT pattern, simply rejecting all steps of negative samples is not sound, potentially undermining beneficial reasoning steps. PPO (Schulman et al., 2017) attempts token-level credit assignment through value model. Nevertheless, its effectiveness is limited by the exponentially growing action space under the long CoT paradigm, requiring substantial data for accurate token-level value estimation.

To better utilize negative samples, we propose a fine-grained offline RL framework named Behavior Constrained Policy Gradient with Negative Sample Augmentation (BCPG-NSA). Our framework first employs a semantic segmentation model to divide negative samples into multiple steps. These steps are then evaluated for correctness using an LLM judger and/or a process reward model (PRM) (Zheng et al., 2024). Finally, we introduce a novel token-level policy optimization objective that enables fine-grained leveraging of negative samples, where the penalization for valuable steps is reduced or even reversed to encourage their generation, with this adjustment strength controlled by the mining coefficient.

Experimental results demonstrate that BCPG-NSA achieves the best performance and improved sample efficiency compared to baselines on challenging math reasoning benchmarks AIME24 and AIME25 and o.o.d coding benchmark LiveCodeBench. In addition, our ablation studies show that the consensus-based LLM-PRM annotation approach yields the best results compared to LLM-only and PRM-only approaches, indicating the importance of precise step correctness assessment in negative samples. We also demonstrate that BCPG-NSA is robust across different mining coefficient values, maintains stability over extended training epochs, and scales effectively to multiple iterations. Our contributions are summarized as follows:

- To the best of our knowledge, we are among the first to empirically validate the value of negative samples through experimental analysis and case studies, and propose mining correct reasoning steps from these samples to enhance long CoT reasoning.
- We introduce BCPG-NSA, an effective offline RL training framework that integrates reasoning step segmentation, consensus-based LLM-PRM annotation, and a policy optimization objective with negative sample augmentation. Across several challenging math and coding benchmarks, BCPG-NSA achieves improved performance and sample efficiency compared with baselines.
- We conduct extensive analyses to demonstrate the robustness and scalability of BCPG-NSA, providing insights into its effectiveness under various conditions.

2 Related Work

Long Chain-of-Thought Reasoning Language Models LLMs have demonstrated remarkable reasoning capabilities in complex tasks. CoT is one of the significant methods to encourage and enhance the reasoning ability of LLMs, which guides LLMs to break down problems and solve them step by step. OpenAI o1 (OpenAI, 2024) is the first to introduce inference time scaling law, which employs large-scale RL to enable autonomous optimization of CoT during training and overcome challenging tasks by generating more reasoning tokens. Several efforts (Team, 2024a; Guo et al., 2025; Team, 2025; Zhang et al., 2025a; Hu et al., 2025) have successfully replicated the inference time scaling law, demonstrating the powerful capabilities of this new inference paradigm.

Process Reward Models in Mathematical Reasoning Mathematical reasoning in LLMs has made significant strides with the introduction of reward models. Reward models are primarily divided into two categories: Outcome Reward Model (ORM) and Process Reward Model (PRM). ORMs only score the final answer of the LLMs' responses, while PRMs assign scores to each reasoning step, providing granular feedback. Consequently, PRMs can not only guide search (Park et al., 2024b; Zhang et al.) but also offer dense rewards in RL training (Gao et al., 2024).

Benefits from Negative Data Many RL methods for LLMs, such as GRPO (Shao et al., 2024) and GPG (Chu et al., 2025), consider every step in a negative sample incorrect, and use the same strength to push down the likelihood of all tokens in incorrect responses. Prior research on learning from negative samples primarily focus on training data construction. Recent works on DPO (Rafailov et al., 2023; Setlur et al., 2024) proposes constructing training pairs with shared prefixes between positive and negative samples, aiming to improve the model's decision-making at critical intermediate steps. Another research targeting SFT data construction (Wang et al., 2024) additionally add a prefix to indicate whether the current generation is a successful trajectory, helping the model better distinguish between correct and incorrect responses. The most closely related work (Li et al., 2024b) leverages valuable signals from negative samples in short CoT reasoning, through a specialized dual-LoRA framework for model distillation. Under the long CoT paradigm, incorrect and correct steps more frequently alternate throughout the reasoning process. Therefore, fine-grained mining of the value of negative samples is a promising direction.

3 Preliminary Analysis: The Value of Negative Samples

In this section, we investigate two fundamental questions: *1) Can supervised fine-tuning on negative samples bring performance benefits?* and if so, *2) To what extent can these benefits be realized?*

Specifically, we construct two distinct training datasets from the open-source R1 dataset (Yang et al., 2025): `SFT-pos` and `SFT-neg`, to compare their respective performance gains when training the base model Qwen2.5-14B-Base (Team, 2024b). Both datasets share identical prompts and contain an equal number of samples (19,000 each). The key difference lies in their composition: `SFT-pos` consists exclusively of responses with correct final answers, while `SFT-neg` contains only responses with incorrect answers.

	AIME24	AIME25
Qwen2.5-14B-Instruct	10.00	13.33
- trained on `SFT-pos`	52.75	39.42
- trained on `SFT-neg`	41.67	34.00

Table 1: Performance improvement under different training set curations.

We draw two conclusions from the results presented in Table 1. Firstly, fine-tuning on negative samples yields substantial performance gains, with the `SFT-neg` model outperforming the base Qwen2.5-14B-Instruct by 31.67% on AIME24. This suggests the presence of valuable components within incorrect responses. Through human investigation, we find that long CoT models often exhibit self-reflection and propose alternative problem-solving approaches during reasoning. We illustrate a representative example in Figure 1 (with complete details provided in Section A).

Secondly, while `SFT-neg` significantly improves upon the base model, it still underperforms compared to `SFT-pos` by 5-10% across benchmarks, due to the presence of flawed reasoning steps in negative samples. This observation motivates us to design a mechanism that efficiently distinguishes and leverages valuable steps within negative samples to maximize performance gains.

4 Method

Our method BCPG-NSA encompasses three stages: thinking process segmentation, consensus-based step correctness annotation, and policy optimization with negative sample augmentation. The overall framework of BCPG-NSA is shown in Figure 1(b).

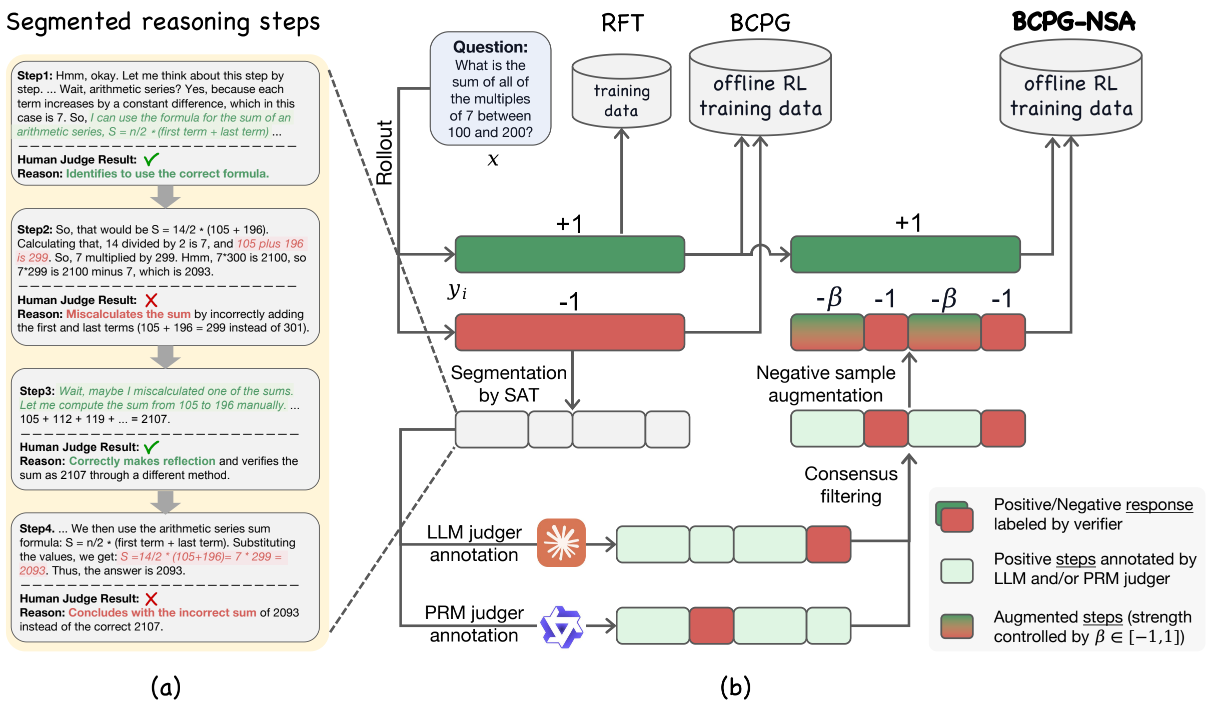
4.1 Segmenting Thinking Process into Steps

To enable a more fine-grained analysis of the negative sample thinking process, we first need to segment the thinking process into multiple steps. Previous approaches to segment the CoT process mainly fall into two categories: rule-based methods, such as splitting using double line breaks (Zhang et al.), and automatic segmentation using LLMs (Zhang et al., 2025b; Zheng et al., 2024). Rule-based methods can lead to semantic discontinuity, such as incomplete reasoning steps or mixing multiple independent logical segments within a single step. On the other hand, automatic segmentation with LLMs may result in content loss after segmentation. Therefore, we choose to use the SAT model (Frohmann et al., 2024). The SAT model not only automatically segments text based on semantics, but also ensures consistency of the text content before and after segmentation. We use a binary search method to find an appropriate segmentation threshold and make the number of steps after segmentation within a reasonable range.

Specifically, for each prompt x, we use a fixed reference policy π_{ref} to generate G responses $\{y_i\}_{i=1}^{G}$. The rollout process of π_{ref} that autoregressively generates the i-th response y_i can be formulated as:

$$\pi_{\text{ref}}(y_i|x) = \prod_{j=1}^{|y_i|} \pi_{\text{ref}}(y_{i,j}|x, y_{i,<j}), \qquad (1)$$

where $y_{i,j}$ denotes the j-th token in response y_i. Given the ground truth y^*, the correctness of y_i is labeled by a verifier (e.g., the well-established `math_verify` (Kydlíek and Face, 2025) for math reasoning tasks), yielding a binary reward $r_i = r(x, y_i, y^*) \in \{0, 1\}$. For negative samples (where $r_i = 0$), each response y_i is segmented by the SAT


Segmented reasoning steps
Question: What is the sum of all of the multiples of 7 between 100 and 200?
x
Rollout
RFT
training data
BCPG
offline RL training data
BCPG-NSA
offline RL training data
+1
y_i
-1
+1
$-\beta$
-1
$-\beta$
-1
Segmentation by SAT
Negative sample augmentation
Consensus filtering
LLM judger annotation
PRM judger annotation
Positive/Negative response labeled by verifier
Positive steps annotated by LLM and/or PRM judger
Augmented steps (strength controlled by $\beta \in [-1,1]$)
(a)
(b)

Step1: Hmm, okay. Let me think about this step by step. ... Wait, arithmetic series? Yes, because each term increases by a constant difference, which in this case is 7. So, *I can use the formula for the sum of an arithmetic series, S = n/2 ∗ (first term + last term)* ...

Human Judge Result: ✓
Reason: **Identifies to use the correct formula.**

Step2: So, that would be S = 14/2 ∗ (105 + 196). Calculating that, 14 divided by 2 is 7, and *105 plus 196 is 299*. So, 7 multiplied by 299. Hmm, 7*300 is 2100, so 7*299 is 2100 minus 7, which is 2093.

Human Judge Result: ✗
Reason: **Miscalculates the sum** by incorrectly adding the first and last terms (105 + 196 = 299 instead of 301).

Step3: *Wait, maybe I miscalculated one of the sums. Let me compute the sum from 105 to 196 manually.* ... 105 + 112 + 119 + ... = 2107.

Human Judge Result: ✓
Reason: **Correctly makes reflection** and verifies the sum as 2107 through a different method.

Step4. ... We then use the arithmetic series sum formula: S = n/2 ∗ (first term + last term). Substituting the values, we get: *S =14/2 * (105+196)= 7 * 299 = 2093*. Thus, the answer is 2093.

Human Judge Result: ✗
Reason: **Concludes with the incorrect sum** of 2093 instead of the correct 2107.

Figure 1: (a): Case study: presence of correct steps (via human judgments) within incorrect response. (b): Overall framework of BCPG-NSA.

model into K consecutive steps:

$$y_i \overset{\text{SAT}}{=} \text{STEP}_{i,1} || \text{STEP}_{i,2} || \cdots || \text{STEP}_{i,K},$$
$$\text{STEP}_{i,k} \triangleq y_{i,\text{start}_k} || \cdots || y_{i,\text{end}_k}, \tag{2}$$

where $||$ is the concatenation operator, and start_k and end_k represent the indices of the starting and ending tokens of the k-th step, respectively.

4.2 Consensus-based Annotation by LLM and PRM Judgers

After performing segmentation on the negative samples, we use a LLM judger and a discriminative PRM to jointly annotate each step in the reasoning process as correct/incorrect.

In designing the LLM judger, we first establish a taxonomy of *context-agnostic* reasoning errors, encompassing calculation mistakes, derivation errors, logical flaws, problem misinterpretations, and similar issues. However, our preliminary experiments reveal that these context-agnostic rules alone are insufficient for comprehensive evaluation. This limitation stems from the distinctive characteristic of long CoT LLMs: their capability to engage in self-reflection and error-correction during the reasoning process, often resulting in alternating sequences of correct and incorrect steps. Consequently, the assessment of a step's correctness cannot be performed independently, but rather depends on its relationship with the preceding context.

Therefore, we augment the annotation criteria with two *context-aware* rules:

Definition 1 (Error Propagation)**.** A reasoning step is classified as *incorrect* if it satisfies two conditions: 1) it follows an incorrect step, or 2) it continues the reasoning based on the previous step without introducing new problem-solving approaches.

Definition 2 (Error Termination)**.** A reasoning step is classified as *correct* if it follows an incorrect step and either: 1) successfully rectifies the previous error, or 2) introduces an alternative problem-solving approach.

Formally, we inject these annotation criteria into the prompt template (see Figure 6) and instruct the LLM judger $\phi^{\text{LLM}}(\cdot)$ to assess the correctness of each step:

$$I_{i,k}^{\text{LLM}} = \phi^{\text{LLM}}(x, \text{STEP}_{i,:k}) \in \{0, 1\}, \tag{3}$$

where $I_{i,k}^{\text{LLM}} = 1$ denotes $\text{STEP}_{i,k}$ is annotated as correct.

Similarly, we use a discriminative PRM $\phi^{\text{PRM}}(\cdot)$ to annotate each reasoning step. The PRM first predicts the score $\sigma_{i,k}$ for $\text{STEP}_{i,k}$, and then applies a threshold λ to determine the annotation outcome:

$$\sigma_{i,k} = \phi^{\text{PRM}}(x, \text{STEP}_{i,:k}) \in [0,1], \\ I_{i,k}^{\text{PRM}} = \mathbf{1}_{\sigma_{i,k}>\lambda} \in \{0,1\}, \tag{4}$$

where $\mathbf{1}(\cdot)$ is the indicator function. In our experiments, λ is determined through grid search, with the selection criterion being to maximize the consistency between LLM and PRM annotating results.

Finally, we introduce a consensus filtering approach, in which a reasoning step is considered correct only when both the LLM judger and PRM deem it correct:

$$I_{i,k} = I_{i,k}^{\text{LLM}} \wedge I_{i,k}^{\text{PRM}}. \tag{5}$$

4.3 Policy Optimization with Negative Sample Augmentation

Drawing from classical offline RL work (Park et al., 2024a), we propose a novel objective function that combines policy improvement with behavior constraint (i.e., KL regularization), while introducing a mechanism to leverage valuable components within negative samples. Our objective function BCPG-NSA is formulated as follows:

$$J_{\text{BCPG-NSA}}(\pi_\theta) = \\ \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\Bigg[\sum_{j=1}^{|y_i|}\log\pi_\theta(y_{i,j}|x,y_{i,<j})\beta_{i,j}(r_i-\overline{r}) \\ -\frac{\tau}{2}\Big(\log\frac{\pi_\theta(y_i|x)}{\pi_{\text{ref}}(y_i|x)}\Big)^2\Bigg], \tag{6}$$

where $\overline{r} = \text{mean}(\{r_i\}_{i=1}^{G})$ is the average reward for the group, τ is the behavior constraint factor. Notably, the value of $\beta_{i,j}$ is given by

$$\beta_{i,j} = \begin{cases} \beta & \text{if } y_{i,j} \in \text{STEP}_{i,k} \text{ and } I_{i,k}=1 \\ & \quad \text{and } r_i = 0 \\ 1, & \text{otherwise,} \end{cases} \tag{7}$$

where the “if” condition indicates that token $y_{i,j}$ appears in a correct step within an incorrect response. The mining coefficient $\beta \in [-1, 1]$ is a hyperparameter that controls the degree of augmentation for correct steps within negative samples. The smaller values of β indicate stronger augmentation of correct steps in negative samples, and intuitively, the penalization for valuable tokens is reduced (when $\beta \in [0, 1)$) or even reversed to encourage their generation (when $\beta < 0$). Specifically, when $\beta = 1$, BCPG-NSA reduces to vanilla BCPG, which penalizes all tokens in negative samples equally. In this case, the objective function resembles the loss formulation of Kimi k1.5 (Team, 2025). A detailed procedure of BCPG-NSA is illustrated in Algorithm 1.

Algorithm 1 BCPG-NSA (single iteration)

```
Input: Reference model $\pi_{\text{ref}}$
/* Offline training data construction */
for $x$ in prompt set do
    Rollout by Equation (1)
    if $r_i = 0$ then
        Response segmentation by Equation (2)
        Step annotation by Equation (3) (4) (5)
    end if
end for
/* Training */
Initialize policy model $\pi_\theta \leftarrow \pi_{\text{ref}}$, mining coefficient $\beta$
for epoch in $1, 2, \cdots$, Epochs do
    Update $\pi_\theta$ by Equation (6)
end for
Output: $\pi_\theta$
```

5 Experiments

5.1 Experiment Setting

5.1.1 Training Dataset Construction

For the offline RL training dataset, we build the prompt seed set based on open-source data collected by the Open-Reasoner-Zero project (Hu et al., 2025), comprising AIME (up to 2023), OpenR1-Math-220k (Ben Allal et al., 2025) and various other open-source datasets (Li et al., 2024a; Lambert et al., 2025; Hendrycks et al., 2021). During data filtering, we exclude multiple-choice and true/false questions to prevent cases where the model might occasionally derive the correct answer through incorrect reasoning steps, which could result in false positives and introduce noise into the training process.

Subsequently, we employ DeepSeek-R1-Distill-Qwen-14B (DS-R1-14B) to generate responses. For each question, we sample 32 responses to ensure sufficient coverage of the response space. We set the maximum response length as 22,000 tokens. The sampling temperature is set to 0.7 to maintain a balance between response diversity and quality.

After generation, we use the open-source verifier tool `math_verify` (Kydlíek and Face, 2025) to check whether the model’s final answer is cor-

Question Count	Total Samples	Negative Samples	Total Tokens
2069	66208	14896	470M

Table 2: Statistics of the offline RL dataset.

rect, and then remove questions for which all responses are either entirely correct or entirely incorrect, following (Wen et al., 2025). After filtering, the statistics of our final offline RL dataset are summarized in Table 2.

5.1.2 Evaluation Details

For evaluation, we choose the highly challenging benchmarks AIME24 (MAA, 2024) and AIME25, along with MATH500 (Hendrycks et al., 2021), to demonstrate the model's mathematical reasoning performance. We also incorporate LiveCodeBench (Jain et al., 2024) (2024/8/1 - 2025/2/1) to show the generalization capabilities to coding tasks. Following DeepSeek-AI (Guo et al., 2025), long CoT models are commonly deployed with a sampling temperature. In our evaluation, we set the temperature to 0.7 (identical to the temperature used in rollout). We report the pass@1 averaged over 40 runs on AIME24 and AIME25. For MATH500 and LiveCodeBench, we average over 10 runs, as these benchmarks exhibit relatively small variance between test runs. This ensures statistical robustness and mitigates randomness in sampling, better reflecting the model's true capabilities.

5.1.3 Models

Base Model. We initialize our training from the DS-R1-14B. The reasons for choosing it as the starting model are as follows: 1) DS-R1-14B has undergone large-scale SFT training and exhibits a stable long CoT pattern with frequent alternation between correct and incorrect steps, making it an ideal starting model to validate the effectiveness of NSA. 2) Based on R1 results (Guo et al., 2025), the 14B model achieves comparable performance to the 32B variant, offering more generalizable findings than the 7B model while being more computationally efficient than the 32B model.

LLM Judger. We employ Claude-3.7-Sonnet(thinking) (Anthropic, 2025), one of the latest slow-thinking models, as the judger model.

PRM. We utilize Qwen-Math-PRM-7B (Zhang et al., 2025b) as the PRM for step-wise annotation. This choice is motivated by its outstanding performance on the ProcessBench (Zheng et al., 2024) Benchmark, demonstrating its robust capability in process evaluation tasks. We set the threshold λ as 0.6.

5.1.4 Baselines

We choose the following methods as baselines:

- RFT (Yuan et al., 2023): It exclusively utilizes positive samples from the offline RL dataset and updates the model parameters through SFT loss.
- DPO (Rafailov et al., 2023): It is a prominent approach in offline RL settings and directly optimizes the preference objectives without explicit reward modeling. This method has demonstrated remarkable effectiveness in preference-based learning tasks.
- TOPR (Roux et al., 2025): As an offline RL variant, it combines truncated importance sampling for negative samples with RFT-style optimization for positive samples, and removes KL regularization.
- GRPO-offline (Shao et al., 2024): It applies the GRPO loss consistently throughout the offline RL training process, eliminates the need for periodic online data resampling.

The training details of all algorithms can be found in Section C.

5.2 Main Results

Negative samples play a crucial role in enhancing the performance of long CoT model. As shown in Table 3, all offline RL methods consistently outperform RFT. Notably, RFT performs even worse than the original DS-R1-14B, exhibiting a significant 7.5% drop on AIME24. These results indicate that positive samples alone are insufficient to improve a distilled model that has already undergone SFT on a large-scale dataset. The negative gradient from negative samples can reduce the probability of wrong reasoning content, which is crucial for advancing model performance in long CoT reasoning. We hypothesize that the reason for the decline in RFT performance is that a substantial proportion of questions in our training dataset have already been utilized in the distillation phase. For the same questions, the answers generated by DS-R1-14B are not as good as

	DS-R1-14B	RFT	DPO	TOPR	GPRO-offline	BCPG	BCPG-NSA
AIME24	70.58	62.92	69.83	69.75	70.50	70.50	**72.17**
AIME25	49.58	50.75	49.00	52.67	50.90	52.00	**54.42**
MATH500	91.80	92.20	92.88	93.12	92.08	**93.98**	93.36
LiveCodeBench	52.40	52.69	51.61	52.90	52.97	53.26	**53.84**
Average	66.09	64.64	65.83	67.11	66.61	67.44	**68.45**

Table 3: Evaluation results of BCPG-NSA and baselines on different benchmarks (β is set to 0.5 for BCPG-NSA).

those generated by R1. Therefore, during RFT, the model memorizes the poorer responses and forgets the better ones from the distillation phase.

Vanilla BCPG sets a competitive foundation. Despite its simplicity, BCPG achieves a 1.4% average improvement across all benchmarks compared to DS-R1-14B, outperforming other offline RL methods. Given its superior performance, we select BCPG as the foundation to validate the effectiveness of NSA.

Negative sample augmentation leads to significant performance gain. Compared to BCPG, BCPG-NSA achieves a remarkable 2.5% improvement on AIME25 and a 1% improvement on average performance. Since DS-R1-14B already performs very well on the AIME24 benchmark, previous work (Wen et al., 2025) has shown that achieving further improvements on AIME24 is quite challenging. However, our BCPG-NSA algorithm successfully achieves an accuracy exceeding 72%. These results validate that the correct steps within negative samples are of great value and can significantly enhance the models reasoning ability.

6 Ablation Studies

6.1 Different Annotation Methods

In this section, we investigate the impact of the annotation quality on the performance of BCPG-NSA. We conduct experiments where negative samples are annotated using PRM only and the LLM judger only, respectively, and compare with the performance obtained by consensus filtering that integrates both LLM and PRM annotations. Table 4 shows the number of tokens in correct steps and incorrect steps among negative samples under different annotation methods. In all three experiments mentioned above, β is set to 0.5.

Results are demonstrated in Table 5. Compared to the base model DS-R1-14B and the vanilla BCPG, negative sample augmentation consistently improves performance across all 3 annotation methods, demonstrating the robustness of NSA's benefits regardless of the choice of judgers. Moreover, the LLM-PRM approach, despite mining the smallest number of correct tokens (only 26M in total) from negative samples, achieves the best performance. We hypothesize that the consensus-based filtering implements more stringent selection criteria, ensuring higher quality of retained correct steps and avoiding potential false positives from individual judgers.

	Correct tokens	Incorrect tokens
PRM-only	38M	80M
LLM-only	65M	53M
LLM-PRM	26M	92M

Table 4: Number of tokens in correct/incorrect steps of negative samples under different annotation methods.

	AIME24	AIME25	Avg.
DS-R1-14B	70.58	49.58	60.08
BCPG	70.50	52.00	61.25
PRM-only	70.58	**55.58**	63.08
LLM-only	**72.17**	53.33	62.75
LLM-PRM	**72.17**	54.42	**63.30**

Table 5: Performance under different annotation methods.

6.2 Different Values of Mining Coefficient

Mining coefficient β is the key hyperparameter in NSA, used to control the strength of the augmentation. Therefore, we test the performance of the model under different β values within the range $[-1, 1]$ to demonstrate the robustness of NSA. When $\beta = 1$, BCPG-NSA reduces to the vanilla BCPG method.

Results in Figure 2 demonstrate that: 1) As β decreases (indicating more aggressive negative sample augmentation), BCPG-NSA's performance ex-

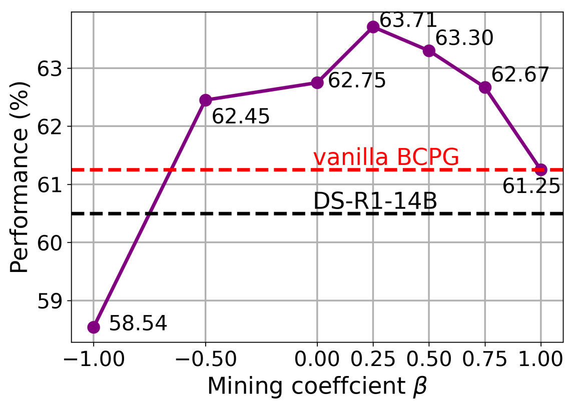


Figure 2: Average performance of AIME24 and AIME25 across different β values.

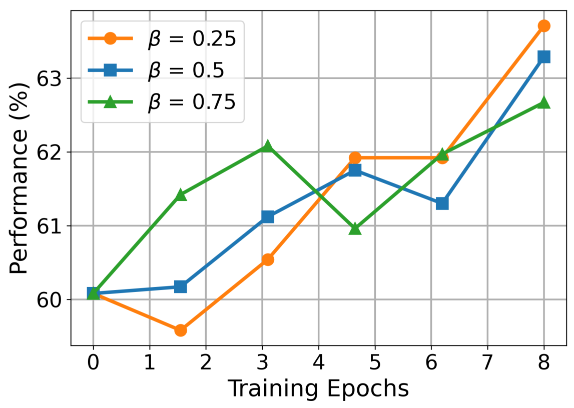


Figure 3: Average performance of AIME24 and AIME25 under different update steps (single iteration).

hibits an initial increase followed by a decline. 2) BCPG-NSA outperforms vanilla BCPG across a wide range of β values, including at the relatively aggressive setting of $\beta = -0.5$. This robust performance across different β values suggests that NSA is not overly sensitive to this hyperparameter and can consistently deliver performance improvements.

7 Further Analysis

7.1 Training Dynamics

In offline RL, the data in the training set is often used multiple times for updates. Therefore, we track the average performance of AIME24 and AIME25 under different update steps during training. We conduct experiments with $\beta \in [0.25, 0.5, 0.75]$. The experimental results in Table 3 show that as the number of training epochs increases, the models performance improves steadily. Therefore, conducting multiple rounds of updates is essential to fully leverage the rollout data and achieve higher sample efficiency.

7.2 The Performance of Multiple Iterations

To investigate the scalability of our proposed method when extended to multiple training iterations, we further evaluate both BCPG-NSA and BCPG in a second iteration. Given the enhanced model capabilities after the first iteration, we sample 1,000 problems from the 13k hard dataset (Hu et al., 2025) derived from the Open-Reasoner-Zero project for the second iteration. Using the model checkpoint from the end of iteration 1, we perform rollouts on these problems and generate 32 responses per problem. As shown in Table 6, BCPG-NSA achieves an additional 1.3% improvement on AIME25 and obtains best average performance in iteration 2, demonstrating its ability to continuously benefit from multiple iterations. Notably, BCPG's performance in iteration 2 remains below that of BCPG-NSA in iteration 1, further validating the effectiveness of NSA.

	AIME24	AIME25	Avg.
DS-R1-14B	70.58	49.58	60.08
BCPG (iteration 1)	70.50	52.00	61.25
BCPG-NSA (iteration 1)	**72.17**	54.42	63.30
BCPG (iteration 2)	71.42	53.42	62.42
BCPG-NSA (iteration 2)	72.00	**55.75**	**63.88**

Table 6: Performance under multiple iterations.

8 Conclusion

In this paper, we present BCPG-NSA, a novel offline RL framework designed to effectively leverage valuable components within negative samples for LLMs' long CoT reasoning. Unlike existing methods that either discard negative samples or apply equal penalization, our approach enables fine-grained treatment of negative samples through three key components: semantic step segmentation, consensus-based step correctness assessment, and policy optimization with negative sample augmentation. Experimental results on challenging reasoning benchmarks demonstrate that BCPG-NSA achieves superior performance, and exhibits great robustness and scalability when extended to multiple iterations.

9 Limitations

Although our proposed negative sample augmentation framework has demonstrated its effectiveness on multiple challenging mathematical and code reasoning benchmarks, there are still several limitations to be addressed. 1) More fine-grained credit assignment. Currently, our method assigns equal credit to all correct steps in negative samples. A potential improvement would be to enable LLM or PRM judger to output continuous values instead of binary decisions, which could then be used to compute a weighted final score, allowing for more precise credit assignment for each step. 2) While the consensus-based LLM-PRM annotation method achieves strong performance, the annotation process could be optimized by training a dedicated model on the labeled data, potentially improving both efficiency and speed. These improvements will be explored in future work.

References

Wasi Uddin Ahmad, Sean Narenthiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. 2025. Opencodereasoning: Advancing data distillation for competitive coding. *arXiv preprint arXiv:2504.01943*.

Anthropic. 2025. Claude 3.7 sonnet. Accessed: 2025.

Loubna Ben Allal, Lewis Tunstall, Anton Lozhkov, Elie Bakouch, Guilherme Penedo, Gabriel Martín Blázquez, and Hynek Kydlicek. 2025. Open r1: Evaluating llms on uncontaminated math competitions. https://huggingface.co/blog/open-r1. Online resource.

Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. 2025. Gpg: A simple and strong reinforcement learning baseline for model reasoning. *arXiv preprint arXiv:2504.02546*.

Mehdi Fatemi, Banafsheh Rafiee, Mingjie Tang, and Kartik Talamadupula. 2025. Concise reasoning via reinforcement learning. *arXiv preprint arXiv:2504.05185*.

Markus Frohmann, Igor Sterner, Ivan Vulić, Benjamin Minixhofer, and Markus Schedl. 2024. Segment any text: A universal approach for robust, efficient and adaptable sentence segmentation. *arXiv preprint arXiv:2406.16678*.

Jiaxuan Gao, Shusheng Xu, Wenjie Ye, Weilin Liu, Chuyi He, Wei Fu, Zhiyu Mei, Guangju Wang, and Yi Wu. 2024. On designing effective rl reward at training time for llm reasoning. *arXiv preprint arXiv:2410.15115*.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. 2025. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Hynek Kydlíek and Hugging Face. 2025. Math-verify. https://github.com/huggingface/Math-Verify.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. 2025. Tulu 3: Pushing frontiers in open language model post-training. *Preprint*, arXiv:2411.15124.

Dacheng Li, Shiyi Cao, Tyler Griggs, Shu Liu, Xiangxi Mo, Eric Tang, Sumanth Hegde, Kourosh Hakhamaneshi, Shishir G Patil, Matei Zaharia, et al. 2025. Llms can easily learn to reason from demonstrations structure, not content, is what matters! *arXiv preprint arXiv:2502.07374*.

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

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Bin Sun, Xinglin Wang, Heda Wang, and Kan Li. 2024b. Turning dust into gold: Distilling complex reasoning capabilities from llms by leveraging negative data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18591–18599.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice

Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. 2025. Deepcoder: A fully open-source 14b coder at o3-mini level. https://pretty-radio-b75.notion.site/DeepCoder-A-Fully-Open-Source-14B-Coder-at-O3-mini-Level-1cf81902c14680b3bee5eb349a512a51. Notion Blog.

MAA. 2024. American invitational mathematics examination - aime.

Yingqian Min, Zhipeng Chen, Jinhao Jiang, Jie Chen, Jia Deng, Yiwen Hu, Yiru Tang, Jiapeng Wang, Xiaoxue Cheng, Huatong Song, et al. 2024. Imitate, explore, and self-improve: A reproduction report on slow-thinking reasoning systems. *arXiv preprint arXiv:2412.09413*.

OpenAI. 2024. Learning to reason with llms. https://openai.com/index/learning-to-reason-with-llms/. Accessed: Month Day, Year.

Seohong Park, Kevin Frans, Sergey Levine, and Aviral Kumar. 2024a. Is value learning really the main bottleneck in offline rl? *arXiv preprint arXiv:2406.09329*.

Sungjin Park, Xiao Liu, Yeyun Gong, and Edward Choi. 2024b. Ensembling large language models with process reward-guided tree search for better complex reasoning. *arXiv preprint arXiv:2412.15797*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741.

Nicolas Le Roux, Marc G Bellemare, Jonathan Lebensold, Arnaud Bergeron, Joshua Greaves, Alex Fréchette, Carolyne Pelletier, Eric Thibodeau-Laufer, Sándor Toth, and Sam Work. 2025. Tapered off-policy reinforce: Stable and efficient reinforcement learning for llms. *arXiv preprint arXiv:2503.14286*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. 2024. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *Advances in Neural Information Processing Systems*, 37:43000–43031.

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

Kimi Team. 2025. Kimi k1.5: Scaling reinforcement learning with llms.

Qwen Team. 2024a. Qwen team. qwq: Reflect deeply on the boundaries of the unknown, 2024b.

Qwen Team. 2024b. Qwen2.5: A party of foundation models.

Xiaoyu Tian, Sitong Zhao, Haotian Wang, Shuaiting Chen, Yiping Peng, Yunjie Ji, Han Zhao, and Xiangang Li. 2025. Deepdistill: Enhancing llm reasoning capabilities via large-scale difficulty-graded data training. *arXiv preprint arXiv:2504.17565*.

Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. 2024. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *arXiv preprint arXiv:2402.11651*.

Liang Wen, Yunke Cai, Fenrui Xiao, Xin He, Qi An, Zhenyu Duan, Yimin Du, Junchen Liu, Lifu Tang, Xiaowei Lv, et al. 2025. Light-r1: Curriculum sft, dpo and rl for long cot from scratch and beyond. *arXiv preprint arXiv:2503.10460*.

Wenkai Yang, Jingwen Chen, Yankai Lin, and Ji-Rong Wen. 2025. Deepcritic: Deliberate critique with large language models. *arXiv preprint arXiv:2505.00662*.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search, 2024a. *URL https://arxiv. org/abs/2406.03816*.

Xiaojiang Zhang, Jinghui Wang, Zifei Cheng, Wenhao Zhuang, Zheng Lin, Minglei Zhang, Shaojie Wang, Yinghan Cui, Chao Wang, Junyi Peng, et al. 2025a. Srpo: A cross-domain implementation of large-scale reinforcement learning on llm. *arXiv preprint arXiv:2504.14286*.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025b. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*.

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

A Case Study: A negative response that incorporates positive steps

Figure 4 and Figure 5 present a detailed case to more specifically demonstrate the results of segmentation and annotation of the negative samples. Model's response is segmented into 15 steps. The *content* field contains the model's reasoning process in each step. A *score* of 1 and 0 indicate the step is correct and incorrect, respectively. The *reason* field provides the rationale for the scoring.

B Prompt Template for LLM Judger

The prompt template for LLM judger in Section 5.1.3 is shown in Figure 6.

C Training Details

The detailed hyperparameters of the RFT method are shown in Table 7.

Parameter	Value
Max Learning Rate	2.5×10^{-6}
Min Learning Rate	1×10^{-7}
Warmup Fraction	0.01
Epochs	6
Batch Size	64
Seq Length	32k

Table 7: Training hyperparameters of RFT.

For other RL methods, the shared hyperparameter settings are shown in Table 8. All experiments are conducted using 64 NVIDIA H800/H100 GPUs. The training of BCPG-NSA takes approximately 14 hours.

Parameter	Value
Max Learning Rate	5×10^{-7}
Min Learning Rate	2.5×10^{-7}
Epochs	8
Batch Size	512
Seq Length	32k

Table 8: Training hyperparameters of offline RL methods.

For both the BCPG and BCPG-NSA, τ is set to 1×10^{-3}. For the DPO algorithm (Equation 8), we set β_{DPO} to 0.5.

$$J_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}\Bigg[\log\sigma\left(\beta_{\text{DPO}}\left(\log\frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} \right.\right. \tag{8}$$

$$\left.\left. - \log\frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right)\Bigg] \tag{9}$$

For the objective of TOPR algorithm (Equation 10), the upper bound of the clip a is set to 1, and the lower bound b is set to 0.

$$J_{\text{TOPR}}(\pi_\theta) = -\mathbb{E}_{y\sim\pi_{\text{ref}}}\Big[\rho(x,y)\,\log\pi_\theta(y|x)\Big]. \tag{10}$$

In the equation above, $\rho(x,y)$ is defined as:

$$\rho(x,y) = \begin{cases} \left[\frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}\right]_b^a, & \text{if } r(x,y,y^*) < 0 \\ 1, & \text{otherwise.} \end{cases} \tag{11}$$

For the GRPO algorithm (Equation 12), we set $\epsilon = 0.2$ and $\beta_{\text{GRPO}} = 1 \times 10^{-3}$.

$$J_{\text{GRPO}}(\pi_\theta) = -\mathbb{E}_{x\sim D,\{y_i\}_{i=1}^G\sim\pi_{\text{ref}}}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\Bigg(\min\Big(\frac{\pi_\theta(y_i|x)}{\pi_{\text{ref}}(y_i|x)}A_i, \text{clip}\big(\frac{\pi_\theta(y_i|x)}{\pi_{\text{ref}}(y_i|x)}\big), 1-\epsilon, 1+\epsilon\Big)A_i\Bigg) - \beta_{\text{GRPO}}D_{KL}(\pi_\theta||\pi_{\text{ref}})\Bigg] \tag{12}$$

Problem: What is the sum of all of the multiples of 7 between 100 and 200?
Solution:

- Step 1
  - **Content:** Hmm, okay. Let me think about this step by step. Maybe I can start by recalling how to find the sum of an arithmetic series because multiples of 7 should form an arithmetic sequence. Wait, arithmetic series? Yes, because each term increases by a constant difference, which in this case is 7. So, I can use the formula for the sum of an arithmetic series. ... Then, dividing both sides by 7, I get 13 = n - 1. Adding 1 to both sides, n = 14. So, there are 14 terms in this sequence.
  - **Score:** 1.
  - **Reason:** Correctly identifies the first term (105), last term (196), and number of terms (14) in the arithmetic sequence.
- Step 2
  - **Content:** Wait, let me verify that. Let's list out some terms to be sure. Starting at 105: 105, 112, 119, 126, 133, 140, 147, 154, 161, 168, 175, 182, 189, 196. Yeah, that's 14 terms. Okay, so n = 14. Now, plugging these values into the sum formula: $S = n/2 * (\text{first term} + \text{last term})$. So, that would be $S = 14/2 * (105 + 196)$. Calculating that, 14 divided by 2 is 7, and 105 plus 196 is 299. So, 7 multiplied by 299.Hmm, 7*300 is 2100, so 7*299 is 2100 minus 7, which is 2093.Wait, really? ...
  - **Score:** 0.
  - **Reason:** Miscalculates the sum by incorrectly adding the first and last terms (105 + 196 = 299 instead of 301) and incorrectly computes the sum up to 200 using the wrong first term (105 instead of 7).
- Step 3
  - **Content:** Wait, that's a different number. So, which one is correct? ... Alternatively, since all these numbers form an arithmetic progression, I can use the formula again. So, $S = n/2 * (\text{first term} + \text{last term})$. So, n is 14, first is 105, last is 196. So, 14/2 is 7, times (105 + 196) is 7*299=2093. ...
  - **Score:** 0.
  - **Reason:** Continues based on STEP 2s errors (299 instead of 301 and incorrect sum up to 200) without correction or a new correct approach.
- Step 4
  - **Content:** Wait, that's a different number. So, which one is correct? ... Alternatively, since all these numbers form an arithmetic progression, I can use the formula again. So, $S = n/2 * (\text{first term} + \text{last term})$. So, n is 14, first is 105, last is 196. So, 14/2 is 7, times (105 + 196) is 7*299=2093. ...
  - **Score:** 0.
  - **Reason:** Continues based on STEP 2s errors (299 instead of 301 and incorrect sum up to 200) without correction or a new correct approach.

Figure 4: An example of negative response that incorporates positive steps (part 1).

- Step 5
  - **Content:** ... Wait, maybe I miscalculated one of the sums. Let me compute the sum from 105 to 196 manually, but that would be time-consuming, but let me try a few terms. 105 + 112 + 119 + 126 + 133 + 140 + 147 + 154 + 161 + 168 + 175 + 182 + 189 + 196. Let me add them step by step. Start with 105. 105 + 112 = 217. 217 + 119 = 336. 336 + 126 = 462. 462 + 133 = 595. 595 + 140 = 735. 735 + 147 = 882. 882 + 154 = 1036. 1036 + 161 = 1197. 1197 + 168 = 1365. 1365 + 175 = 1540. 1540 + 182 = 1722. 1722 + 189 = 1911. 1911 + 196 = 2107.
  - **Score:** 1.
  - **Reason:** Correctly verifies the sum as 2107 through manual addition and the difference method, despite repeating an earlier incorrect calculation.
- Step 6
  - **Content:** ...... Wait, so the manual addition gives me 2107, same as 2842 - 735.But when I used the arithmetic series formula, I got 2093. So, that's the problem. Why is there a discrepancy? Wait, let me check the arithmetic series formula again. I think the issue is with the number of terms.Wait, the first term is 105, which is 15*7, and the last term is 196, which is 28*7.
  - **Score:** 1.
  - **Reason:** Correctly applies the alternative arithmetic series formula to get 2107, confirming the correct sum.
- Step 7
  - **Content:** Wait, earlier when I did the arithmetic series formula correctly, I think I messed up. Wait, no. If I use $S = n/2 * (\text{first} + \text{last})$, which is $(14/2 \times (105 + 196) = 7 \times 299 = 2093$. But when we I used the correct formula $S = n/2 * (2a + (n - 1)d)$, it gave me 2107. Wait, that's a contradiction. So, what's happening here. Wait, perhaps I made an error in the number of terms. ... So, n=14, first term 105, last term 196. So, the sum is $14/2 * (105 + 196) = 7 * 299 = 2093$ ...
  - **Score:** 0.
  - **Reason:** Reintroduces the incorrect sum (2093) alongside the correct 2107 without resolving the error.
- Step 8-14. A series of deductions based on false premises.
- Step 15
  - **Content:** So, there are 14 terms in this sequence. We then use the arithmetic series sum formula: $S = \frac{n}{2} \times (\text{first term} + \text{last term})$. Substituting the values, we get: $S = \frac{14}{2} \times (105 + 196) = 7 \times 299 = 2093$. Thus, the sum of all multiples of 7 between 100 and 200 is $\boxed{2093}$.
  - **Score:** 0.
  - **Reason:** Concludes with the incorrect sum of 2093 instead of the correct 2107.

Figure 5: An example of negative response that incorporates positive steps (part 2).

You are an expert in evaluating mathematical problem-solving processes. The user will provide a math problem, a step-by-step solution and ground truth. Your tasks are as follows:

1. First, extract a concise final answer (Short Answer) from the ground truth.
2. Then, carefully review the user's step-by-step solution, assigning a score to each step (either 0 or 1). For each step, provide a brief explanation of your judgement result.

Scoring rules:

- If a step contains an explicit error, such as a reasoning error, calculation mistake, logical flaw, or misunderstanding of the problem, it should be scored 0.
- If a step does not contain any errors, score it according to the following rules:
  1. **Error Propagation:** If a previous step contains an error and the current step continues the analysis based on that error without introducing a new, correct approach or making a proper correction, the current step should also be scored 0.
  2. **Error Termination:** If a previous step contains an error, but the current step either corrects the previous mistake or introduces a new and correct approach, the current step should be scored 1. For example:
     - STEP K contains an error.
     - STEP K+1 continues the analysis based on the error.
     - STEP K+2 corrects the previous error or introduces a new and correct approach.

     In this case, STEP K and STEP K+1 should be scored 0, and STEP K+2 should be scored 1.

Your response format should be in json format:

```
[
    {
        "STEP 0": 1(int),
        "Reason": xxxx(str)
    },
    {
         "STEP 1": 1(int),
         "Reason": xxxx(str)
    }
    ...
]
```

Note: When analyzing the solution, remain objective and rational. Do not be misled by the way the user's solution is described.

Figure 6: Prompt template for LLM judger.