SPC: Evolving Self-Play Critic via Adversarial Games for LLM Reasoning

Jiaqi Chen[1] **Bang Zhang**[*] **Ruotian Ma**[2†] **Peisong Wang**[3]
Xiaodan Liang[4] **Zhaopeng Tu**[2] **Xiaolong Li**[2] **Kwan-Yee K. Wong**[1†]
[1]The University of Hong Kong [2]Tencent
[3]Tsinghua University [4]MBZUAI
Project: https://chen-judge.github.io/SPC/

Abstract

Evaluating the step-by-step reliability of large language model (LLM) reasoning, such as Chain-of-Thought, remains challenging due to the difficulty and cost of obtaining high-quality step-level supervision. In this paper, we introduce **S**elf-**P**lay **C**ritic (**SPC**), a novel approach where a critic model evolves its ability to assess reasoning steps through adversarial self-play games, eliminating the need for manual step-level annotation. SPC involves fine-tuning two copies of a base model to play two roles, namely a "sneaky generator" that deliberately produces erroneous steps designed to be difficult to detect, and a "critic" that analyzes the correctness of reasoning steps. These two models engage in an adversarial game in which the generator aims to fool the critic, while the critic model seeks to identify the generator's errors. Using reinforcement learning based on the game outcomes, the models iteratively improve; the winner of each confrontation receives a positive reward and the loser receives a negative reward, driving continuous self-evolution. Experiments on three reasoning process benchmarks (ProcessBench, PRM800K, DeltaBench) demonstrate that our SPC progressively enhances its error detection capabilities (e.g., accuracy increases from 70.8% to 77.7% on ProcessBench) and surpasses strong baselines, including distilled R1 model. Furthermore, SPC can guide the test-time search of diverse LLMs and significantly improve their mathematical reasoning performance on MATH500 and AIME2024, surpassing those guided by state-of-the-art process reward models.

1 Introduction

The Chain-of-Thought (CoT) [1–3] reasoning process, which emerges in the autoregressive generation of large language models (LLMs), has been applied to address a variety of complex tasks [4–13]. Training methods such as Supervised Fine-Tuning (SFT) [14, 15], Reinforcement Learning from Human Feedback (RLHF) [16, 17], and self-play reinforcement learning [18, 19], have demonstrated success in obtaining high-quality CoT. Recently, the popular o1 [20], R1 [21], and QwQ [22] utilize large-scale reinforcement learning for training and employ test-time scaling to generate long CoT, further enhancing their reasoning capabilities. As the CoT generated by LLMs becomes increasingly complex and diverse, it is particularly important to verify the reliability of the reasoning process, analyze the potential errors in reasoning steps, and guide the test-time search to improve the reasoning process [23–31].

A number of verification models have been developed to analyze and evaluate the reasoning process of LLMs. For example, outcome verifiers [25] provide outcome-level validation to rank or reward multiple responses from LLMs. Process verifiers [23, 25], which validate each step in the reasoning

[*]Independent researcher. †Corresponding authors.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

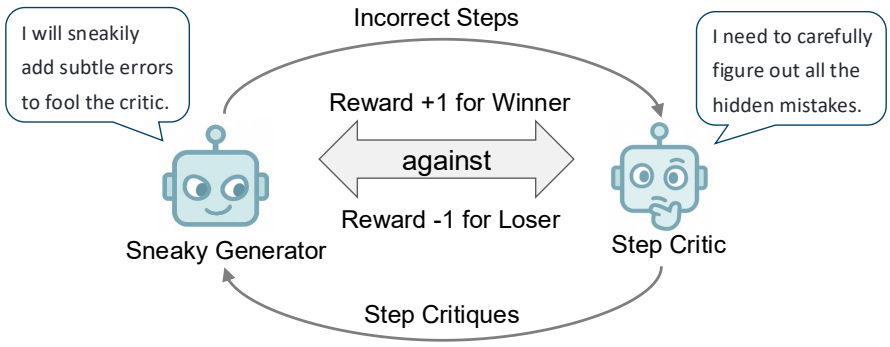

Figure 1: We continuously generate reinforcement training samples for the critic through adversarial games. The sneaky generator aims to create subtle erroneous steps to challenge the critic, while the critic must accurately distinguish between correct and incorrect steps from a mixed input of them. Benefiting from the opposing optimization objectives, both models can evolutionally learn from each other, akin to how humans improve their skills in board games through competition.

process, have proven crucial in recent advances in LLM reasoning [26, 32–34]. However, there are several challenges that limit the development of such step-level approaches. Firstly, while it is relatively simple to extract the final predicted answer and determine the correctness of a solution, determining the correctness of a reasoning step and automatically obtaining well-annotated step data for training a process verifier is much more difficult. Secondly, LLMs are updated rapidly, and heavy human expert annotations on the outputs of specific LLMs may not be applicable to the latest LLMs due to distributional differences. Thirdly, the dataset limited to step correctness annotations restricts the training of a critic model – preventing it from providing substantive feedback and reducing it to merely a scoring mechanism for verification.

In this paper, we introduce a novel **S**elf-**P**lay **C**ritic (**SPC**) to diagnose potential errors and provide valuable critiques for each step in the mathematical reasoning process. Inspired by the self-play framework [19], we propose an adversarial game between a sneaky generator and a critic to continuously generate samples for reinforcement learning, thereby evolving the capabilities of the critic model. Specifically, we first employ supervised fine-tuning to initialize a base model as a sneaky generator, converting correct steps into incorrect steps that can significantly impact the success rate of problem-solving. Concurrently, we initialize an identical base model to play the role of a critic, whose goal is to identify the correctness of these reasoning steps and provide some critiques for them. As shown in Fig. 1, we put these two models in an adversarial game by feeding the incorrect steps successfully generated by the sneaky generator to the critic. Through this adversarial game, we anticipate that the sneaky generator can simulate errors that can practically influence the reasoning of LLMs while remaining difficult for the critic to detect. On the other hand, the critic is expected to gradually address its shortcomings and improve its ability to catch all errors in the reasoning steps. Benefiting from this design, we continuously generate positive/negative samples from different LLMs for reinforcement learning without the need for additional human annotations, facilitating the iterative evolution of a critic model which can provide valuable step critiques.

Extensive experiments have been conducted to validate the effectiveness of our proposed self-play critic. After one round of supervised fine-tuning on Qwen2.5-7B-Instruct and two rounds of iterative reinforcement fine-tuning, our SPC has shown continuously evolving performance on three human-annotated reasoning process assessment benchmarks (ProcessBench [27], PRM800K [23] and DeltaBench [35]). For instance, the average accuracy of SPC on PRM800K has gradually improved from 71.0% to 75.8%, surpassing the 71.4% performance of the same-sized distilled model of R1 [21]. We further introduce a new approach to utilize our tailored critic model, wherein the critic predicts the correctness of each step during LLMs' test-time search. This allows the LLM to promptly abandon incorrect steps and regenerate new steps, rather than waiting until the entire solutions are generated and then scoring them using verifiers. Experiments on MATH500 [36] and AIME2024 [37] indicate that SPC can enhance mathematical reasoning for three different types of LLMs, including popular Llama [38], Qwen [12], and distilled R1 [21] with long CoT reasoning process.

2 Related Work

LLM Reasoning Powerful large language models (LLMs) [4, 5, 7–13] are becoming increasingly adept at constructing Chain-of-Thought (CoT) to tackle complex reasoning tasks, such as solving math problems and code generation. The recently popular o1 [20], R1 [21], and QwQ [22] models are further equipped with exceptional deep thinking capabilities, allowing them to construct long CoT during inference to decompose complex tasks and even perform extensive self-critique and self-correction [39–41]. However, fine-grained analyses in a recent research [35] indicate that the effective proportion of self-critique in these long CoT is still very low, and biases exist in the self-critique of their own reasoning processes. It is therefore necessary to have a simple external critic for assessing the reasoning steps of various LLMs, providing step-level critiques.

Verification and Critique for LLM Verifiers [23–27] can enhance reasoning performance by ranking or integrating multiple responses generated by LLMs during inference. Additionally, they can also provide more accurate rewards during training to guide the optimization of LLMs. Verifiers can be categorized into two types, namely outcome reward models (ORMs) and process reward models (PRMs). ORMs provide solution-level scores for the entire problem-solving process, whereas PRMs assign step-level scores to each reasoning step, which can be aggregated to produce a more accurate solution-level score. Recent works [28–30, 42] propose critic models for verification, arguing that scalar scores have limited ability in evaluating the outputs of LLMs. In contrast, feedback in natural language form can activate the thinking capabilities of LLMs, resulting in more reliable critiques to represent the correctness of reasoning. In this work, we explore how to analyze the correctness of the current step and provide step-level critiques based on partial reasoning steps.

Self-Play Self-play [43, 44] is a method in reinforcement learning where an agent interacts with several copies of itself in an environment to learn specific actions. A significant advancement in self-play is demonstrated by AlphaGo [45] and AlphaZero [46], which greatly surpass human champions in the game of Go, without the need for human knowledge in training. Recent studies apply self-play to LLM alignment and enhancement [18, 19, 47–50]. For example, Kirchner et al. [19] proposed a solution-level game between a powerful prover and a weak scoring verifier to enhance the legibility of the LLM, though resulting in performance degradation. Cheng et al. [47] introduced a Taboo language game between an attacker and a defender to improve LLMs' reasoning abilities. In this paper, we design an adversarial game to generate data for training a step-level critic, which provides correctness analysis for the reasoning steps of LLMs.

3 Methodology

3.1 Overview

Training a step-level critic requires a large amount of data annotated with step correctness. However, collecting step-level data presents considerable challenges. First, identifying and annotating the reasoning errors of powerful LLMs requires professionals with relevant expertise. Second, LLMs are rapidly updated, and the labor-intensive annotations may become outdated and inapplicable to the latest LLMs due to distributional shifts. Third, there is no definite answer for each step, complicating the definition of "incorrect" and the automation of the annotation process.

In this work, we design a self-play framework to enable the self-evolution of step-level critic by automatically producing step-level annotation through an adversarial game. As shown in Fig. 2, our framework involves two opposing models, i.e., a sneaky generator S and a step-level critic C.

Sneaky generator converts correct reasoning steps from LLMs into incorrect ones, automating the creation of numerous steps with potential errors. Its goal is to generate sneaky steps that not only decrease the reasoning success rate of LLMs but also deceive the critic (i.e., the critic fails to detect the erroneous steps). Specifically, given a problem p and a correct partial reasoning trajectory $\tau_{:k} = (t_1, t_2, ...t_k)$ produced by an LLM solver, the sneaky generator S converts the last correct step t_k^c into a *candidate sneaky step* $t_k^i = S(p, \tau_{:k-1}, t_k^c)$. This candidate becomes a *valid sneaky step* $\underline{t_k^i}$ if it significantly impacts the solver's success rate in the subsequent completion from this step. If the sneaky step is invalid or the critic detects the errors in a valid sneaky step, the sneaky generator receives a negative reward. Conversely, if the critic fails to detect the errors in a valid

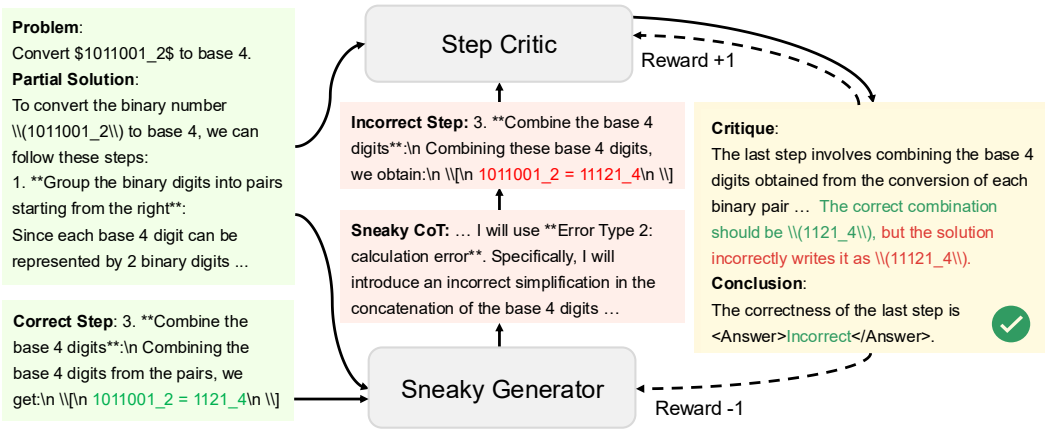

Figure 2: The framework of our proposed SPC. We randomly select a correct step along with the partial solution before that step and feed them into the sneaky generator, which first selects one of the predefined error types and then converts the correct step into an incorrect step. The successfully generated incorrect step is then fed to the critic for error detection. If the critic successfully identifies the error, it receives a reward of +1, while the sneaky generator incurs a reward of -1. If the critic is deceived, the critic and sneaky generator are rewarded -1 and +1, respectively.

sneaky step, the sneaky generator receives a positive reward. By automatically evaluating the success of the generated sneaky steps, we then employ reinforcement learning to enable the self-evolution of the sneaky generator.

Step Critic aims to identify all potential errors in the reasoning steps of LLMs. In each iteration of the adversarial game, the critic's role is to detect all error steps generated by the sneaky generator. Specifically, given a partial reasoning trajectory $\tau_{:k-1} = (t_1, t_2, ...t_{k-1})$ and a valid sneaky step $\overline{t_k^i}$ produced by the sneaky generator S, the critic C is expected to identify $\overline{t_k^i}$ by generating a step-level critique. The success or failure of detecting the sneaky step determines the critic's rewards, allowing continuous optimization through reinforcement learning.

Overall, these two models have opposing objectives, allowing them to evolve through adversarial self-play. In the following sections, we explain how to initialize these models and continuously generate positive and negative samples for reinforcement learning through adversarial games.

3.2 Initializing Sneaky Generator

To initialize the sneaky generator S_0, we train the base model Qwen2.5-7B-Instruct [12] using Supervised Fine-Tuning (SFT) to equip it with the fundamental capability to generate incorrect steps. To ensure the accuracy of the initialization data, we use correct-incorrect step pairs from PRM800K [23] to construct an error step transformation process. Specifically, we extract correct-incorrect step pairs $< t_k^c, t_k^i >$ with the same problem p and partial solution $\tau_{:k-1} = (t_1, t_2, ...t_{k-1})$ (the steps preceding the extracted pairs) from PRM800K. We next prompt GPT-4 to create a chain-of-thought transformation $\mathcal{T}_{\text{CoT}}(t_k^c) \rightarrow t_k^i$ by first selecting an error type from five predefined common error types (see Sec. B) and then performing a detailed transformation. This process results in a transformation behavior cloning dataset $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}_{bc}^S$, where $\mathbf{x} = (p, \tau_{:k-1}, t_k^c)$ as input, $\mathbf{y} = \mathcal{T}_{\text{CoT}}(t_k^c) \rightarrow t_k^i$ as output. We then finetune Qwen2.5-7B-Instruct on dataset $\mathcal{D}_{bc}^S$ to obtain a policy π_θ for the initial sneaky generator S_0 using the SFT loss:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}_{bc}^S}[\log \pi_\theta(\mathbf{y}|\mathbf{x})]. \tag{1}$$

Automated Validation for Sneaky Generator To form an adversarial game, we need to annotate the generated steps and feed actual incorrect steps to the critic model. However, existing LLM-as-a-Judge methods [35, 51] inevitably introduce bias, while the manual annotation is excessively labor-intensive. We therefore propose evaluating the impact of different steps on the problem-solving success rate to ascertain whether a sneaky step can be considered incorrect. Concretely, based on a

correct solution generated by an open-source LLM, we first sample an original step and transform it into a sneaky step using the sneaky generator. We subsequently use the same LLM to complete the entire reasoning process after the original/sneaky steps, and this is repeated N times. If the original step achieves a relatively high success rate while the sneaky step results in a significantly lower success rate, we consider this pair of steps to represent correct and incorrect steps, respectively. In our experiment, we adopt a strict criterion to ensure data quality. If the original step achieves a success rate greater than or equal to 75%, while the sneaky step results in a success rate of 0%, we then collect this pair of steps for subsequent adversarial games.

3.3 Initializing Step Critic

Based on the results from ProcessBench [27], reasoning models such as QwQ [22] and distilled R1 models [21] outperform non-reasoning models such as GPT when serving as critic models. However, the lengthy reasoning process in R1 leads to slow and redundant model generation, and its instruction-following capability is relatively poor, often failing to produce a concise critique with a definite conclusion about the correctness of a step. We therefore combine the strengths of both types of models when initializing the critic.

Specifically, we prompt DeepSeek-R1-Distill-Qwen-7B as a critic, taking problem p, partial solutions $\tau_{:k-1}$, and mixed correct and incorrect steps t_k from PRM800K dataset as inputs, to collect long critiques. We then employ GPT-4 to rewrite them into brief standardized critiques Q_t (see Sec. B). Concretely, Q_t contains an analysis of the partial solution and the current last step, as well as a definite conclusion regarding the correctness of the step. This also simplifies the task and facilitates the use of SFT for policy initialization. Additionally, when preparing the training data for the critic, we mix the steps labeled as correct and incorrect in PRM800K at a 1:1 ratio to ensure the critic's capabilities are balanced. We utilize human annotations from PRM800K to filter around 21.8K correctly generated critiques $\overline{Q_t}$. Similarly, we prepare behavior cloning dataset $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}_{bc}^{C}$ for the critic, where $\mathbf{x} = (p, \tau_{:k-1}, t_k)$ as input, and $\mathbf{y} = \overline{Q_t}$ as output. We then finetune the base model using SFT loss (1) to obtain an initial policy C_0 for the critic.

3.4 Adversarial Game

We further reinforce the models' correct behavior and continuously improve their performance, avoiding the limitations related to the scale and distribution of human-annotated PRM800K. Inspired by recent self-play practices [19, 47], we propose a step-level adversarial game between the sneaky generator and step critic, enabling continuous reward generation and self-evolution of the two roles.

In each iteration of the adversarial game, we begin by using LLM solvers to generate a set of original step-by-step solutions for each problem. To ensure data diversity, we employ various LLM solvers from different model families, with sizes ranging from 7B to 32B, thereby enriching the diversity of sample styles. We then design an adversarial game for the two roles based on these solutions. Single steps are randomly selected from solutions for sneaky transformation, and the incorrect steps successfully produced by the sneaky generator are then fed into the critic to generate critiques. In addition to ensuring that the generated step contains an error, we expect the sneaky generator to generate incorrect steps with subtle flaws that can fool and challenge the critic. Meanwhile, the critic should be powerful enough to avoid being misled by any errors and provide an accurate critique.

In this game, we can set the rewards for the sneaky generator and the critic respectively in an adversarial instance as follows:

$$R_{sneaky} = \begin{cases} 1, & \texttt{Sneaky Generator Wins} \\ -1, & \texttt{Sneaky Generator Loses} \end{cases} \tag{2}$$

$$R_{critic} = \begin{cases} 1, & \texttt{Critic Wins} \\ -1, & \texttt{Critic Loses} \end{cases} \tag{3}$$

This opposing optimization goal enables both the sneaky generator and the critic to continuously improve their performance, achieving iterative self-evolution.

3.5 Evolving via Reinforcement Learning

In each iteration, after obtaining positive and negative samples through the adversarial games, we apply offline reinforcement learning to the critic and sneaky generator, respectively, enabling self-improvement of both roles based on the game result. Specifically, we adopt the following optimization objective to achieve stable RL training:

$$\nabla_\theta \hat{\mathcal{L}}(\theta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim\pi_{\text{old}}(\mathbf{y}|\mathbf{x})}\Big[\frac{\pi_\theta(\mathbf{y}|\mathbf{x})}{\pi_{\text{old}}(\mathbf{y}|\mathbf{x})}\cdot \hat{A}^{\pi_{\text{old}}}(\mathbf{x},\mathbf{y})\cdot\nabla_\theta \log \pi_\theta(\mathbf{y}|\mathbf{x})\Big], \tag{4}$$

where π_{old} denotes the policy used to collect the offline dataset, $\frac{\pi_\theta(\mathbf{y}|\mathbf{x})}{\pi_{\text{old}}(\mathbf{y}|\mathbf{x})}$ is the importance ratio, and $\hat{A}^{\pi_{\text{old}}}$ represents the advantage estimation. Inspired by recent RLOO [52] and GRPO [53], we formulate $\hat{A}^{\pi_{\text{old}}} = R(\mathbf{x},\mathbf{y}) - b - \beta\text{KL}[\pi_\theta||\pi_{\text{ref}}]$, where a baseline b (the average reward of all samples) is subtracted for advantage estimation, and a Kullback-Leibler (KL) penalty is added to regularize the policy π_θ and prevent it from deviating too far from the initial policy π_{ref}.

For the sneaky generator, considering that we also need it to generate actual incorrect steps, we treat sneaky steps that fail to affect the problem-solving success rate as negative samples. Additionally, sneaky steps that successfully impact the LLM success rate but do not deceive the critic will also be considered negative samples. Meanwhile, the ones that can both influence the LLM success rate and deceive the critic are considered positive samples. Consequently, our data for training the sneaky generator includes a 1:1:1 ratio of positive samples and two types of negative samples.

As for the critic, we mix some correct steps from correct solutions with some incorrect steps generated by the sneaky generator for the critic to predict. The samples that the critic successfully predicts receive a positive reward, while those that are incorrectly predicted receive a negative reward. Ultimately, positive/negative samples each constitute half of the total samples.

Based on the adversarial game, we apply iterative training to enable continuous evolution of the two roles. Specifically, in each iteration, the newly updated policies re-engage in the adversarial game to generate new data for training, thereby evolving themselves further. Additionally, we observe an interesting phenomenon that more balanced adversarial games contribute to the self-evolution of models. In fact, the initial sneaky generator S_0 is weaker than the initial critic C_0, resulting in an unbalanced win rate. Moreover, S_1 obtained through synchronous iteration is even weaker than C_1. Therefore, we adopt an asymmetric evolution strategy, where S_1 competes against C_0 in a more balanced game to generate the second round of data. This enables C_2 trained in the second round to further improve its performance. Such a strategy is analogous to humans preferring to improve their skills in chess by playing against equally matched opponents. We provide more detailed analyses of the evolving strategies in Sec. 4.3.

3.6 Enhancing LLM Reasoning

Previous process reward models (PRMs) [26, 27] require scoring each step of the fully generated solutions and then integrate all the scores. However, after the first reasoning step error occurs, the LLM should promptly correct the mistake. Continuing to generate more potentially flawed reasoning steps after an erroneous step is unnecessary, and the scores produced are unreliable. In contrast, we propose a new approach that directly employs a critic to assist the LLM in searching for reasoning steps. During testing, we use '`\n\n`' to control the LLM to output one step at a time, allowing the critic to verify the correctness of each step. If the step is correct, the search continues; if incorrect, the LLM is required to regenerate the step (up to five attempts before skipping). Our SPC effectively enhances the reasoning performance of the LLM using this approach.

4 Experiments

4.1 Experimental Settings

Evaluation We adopt PRM800K [23], ProcessBench [27], and DeltaBench [35] that include human annotations of mathematical reasoning steps for evaluation. The original setting of ProcessBench and DeltaBench is to identify the position of the first or all errors in a complete solution. We argue that, in practical scenarios, a critic can enhance reasoning performance by identifying the incorrect

Table 1: Comparison of recall on ProcessBench. We evaluate different models on their ability to assess the correctness of the current step, instead of only predicting the index of the first error in the complete solution. 'Round 0' refers to the initialized critic model.

Models	GSM8K	MATH	Olympiad-Bench	Omni-MATH	Average
Process Reward Models (PRMs)					
Math-Shepherd-PRM-7B [26]	58.0	58.4	68.0	64.1	62.1
Qwen2.5-Math-7B-PRM800K [27]	77.0	72.9	66.9	62.1	69.7
Prompting LLMs as Critic Models					
Llama-3.1-8B-Instruct [10]	59.5	57.7	53.6	53.9	56.2
Llama-3.1-70B-Instruct [10]	67.2	62.8	61.7	61.9	63.4
Qwen2.5-7B-Instruct [12]	64.2	64.0	62.1	60.8	62.8
Qwen2.5-32B-Instruct [12]	76.2	68.1	68.9	63.9	69.3
GPT-4o [6]	75.5	70.5	70.0	64.5	70.1
DeepSeek-R1-Distill-Qwen-7B [21]	79.0	**81.3**	73.4	67.3	75.2
Our Critic Models					
SPC (Round 0)	78.0	74.1	67.8	63.2	70.8
SPC (Round 1)	82.0	80.3	74.8	**70.3**	76.8
SPC (Round 2)	**84.2**	80.8	**76.5**	69.2	**77.7**

step and requiring the LLM to regenerate, with no need to wait for completing all the steps. We therefore extract a 1:1 ratio of correct and erroneous steps from each benchmark, only retain the reasoning process before these steps as a partial solution, and discard the reasoning steps after these steps. Besides, we evaluate the effectiveness of the critic models in assisting LLMs to solve math problems on MATH500 [54] and AIME2024 [37]. More evaluation details are provided in Sec. C.

Baselines Following ProcessBench [27], we primarily evaluate two types of baselines, namely Process Reward Models (PRMs) and prompting LLMs as critic models. For PRMs, we select two representative methods, namely Math-Shepherd [26] and Qwen2.5-Math-7B-PRM800K [27]. Math-Shepherd trains a process reward model through an automated data annotation process and can be utilized to rank multiple outputs or ensemble them to enhance reasoning performance. Qwen2.5-Math-7B-PRM800K is based on the advanced math-specialized model Qwen2.5-Math-7B [55], and is further fine-tuned with the PRM800K dataset, obtaining state-of-the-art performance among PRMs. We also prompt multiple types of LLMs to serve as critic models, using the same prompts as our SPC. Several representative models, including Llama [10], Qwen [12], R1 [21], and GPT-4o [6], are selected as baselines.

4.2 Main Results

Critic Performance on Reasoning Process Benchmarks As shown in Tabs. 1 and 2, we compare our critic models with other baselines on 3 math-related reasoning process benchmarks to evaluate the abilities of predicting step correctness. We can observe that: (1) Our proposed SPC is gradually evolving and achieves state-of-the-art performance among all models. For example, the average performance on ProcessBench has improved from 70.8% to 77.7%, and on DeltaBench from 54.9% to 60.5%. (2) On all benchmarks, our method outperforms the latest PRMs specifically designed for scoring steps. (3) The performance of prompting LLMs as critics is not as good as SPC. Our method outperforms the distilled R1 model with the same size of 7B parameters. (4) Some baselines (PRMs and prompting Llama) have imbalanced recall between correct and error steps, leading to poor harmonic mean, whereas our critic is more balanced. (5) Our critic is trained on short CoT data (from Qwen and Llama), but it is able to generalize to long CoT reasoning steps (e.g., R1 [21] and QwQ [22]) in DeltaBench. In contrast, the two PRMs trained on short CoT show a significant performance decline in DeltaBench, with HarMean scores of only 14.3% and 41.3%, respectively.

The Effectiveness of Guiding Test-Time Search Existing PRMs can enhance performance by ranking the completely generated reasoning steps or by aggregating scores using self-consistency [2, 26]. We apply the proposed SPC to LLM reasoning search, utilizing SPC to check the correctness of each step and regenerating the step if it is incorrect (up to 5 retries). Moreover, SPC can be combined with self-consistency by conducting a majority vote over several independent searches. For a fair

Table 2: Comparison of our SPC with baselines on the test set of PRM800K [23] and DeltaBench [35], where human-annotated correct and erroneous steps are extracted to evaluate the recall of critiques. "Correct" and "Error" represent the recall on correct and erroneous steps, respectively. "Average" denotes their arithmetic average and "HarMean" refers to their harmonic mean.

	PRM800K				DeltaBench			
Models	**Average**	**HarMean**	**Correct**	**Error**	**Average**	**HarMean**	**Correct**	**Error**
Process Reward Models (PRMs)								
Math-Shepherd-PRM-7B [26]	50.0	49.5	55.2	44.8	53.3	14.3	7.69	**98.8**
Qwen2.5-Math-7B-PRM800K [27]	73.6	73.6	74.4	72.8	58.5	41.3	**90.1**	26.8
Prompting LLMs as Critic Models								
Llama-3.1-8B-Instruct [10]	51.9	30.5	18.6	85.2	49.1	6.38	3.30	95.0
Llama-3.1-70B-Instruct [10]	54.6	38.9	25.3	83.9	44.6	20.3	11.7	77.5
Qwen2.5-7B-instruct [12]	52.8	37.2	24.1	81.6	48.2	33.8	21.8	74.7
Qwen2.5-32B-instruct [12]	59.0	50.5	36.6	81.4	44.7	33.0	21.8	67.6
GPT-4o [6]	68.5	68.4	70.3	66.6	49.9	48.7	42.0	57.9
DeepSeek-R1-Distill-Qwen-7B [21]	71.4	71.2	67.3	75.5	50.9	50.6	54.9	46.9
Our Critic Models								
SPC (Round 0)	71.0	70.8	67.8	74.2	54.9	53.5	45.9	64.0
SPC (Round 1)	72.8	70.3	59.4	**86.1**	58.8	57.3	68.4	49.3
SPC (Round 2)	**75.8**	**75.8**	**74.8**	76.9	**60.5**	**59.5**	68.2	52.8

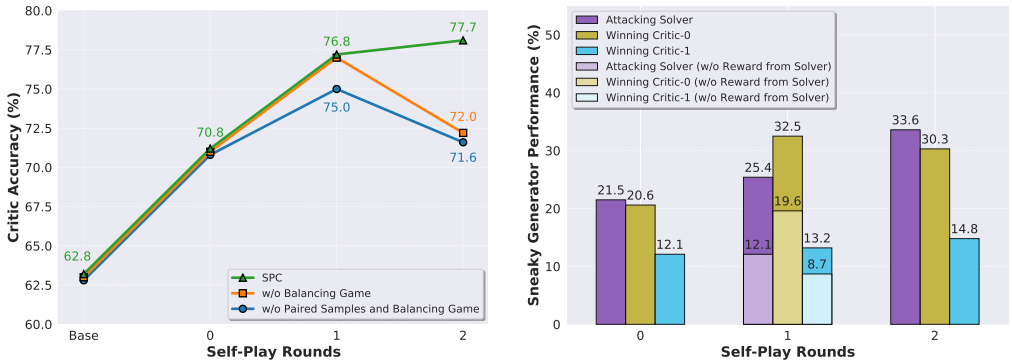

Figure 3: Ablation study of our critic and sneaky generator. Left: The impact of different strategies on evolving critic models. Right: The success rate of sneaky generator in attacking LLM solver and its win rate against round 0 and round 1 critics.

comparison, all methods incorporating self-consistency sample 5 outputs in our experiments. In addition, for experiments without using self-consistency, we run them at least three times and average the results to reduce randomness. As shown in Tab. 3, on two popular benchmarks MATH500 [54] and AIME2024 [37], SPC significantly improves the performance of three types of LLM solvers, and outperforms five baseline verifiers. For instance, using the Qwen Solver at AIME2024, our SPC combined with Self-Consistency achieves a problem-solving accuracy of 23.3%, which is superior to the 16.7% accuracy of Self-Consistency + Qwen2.5-Math-7B-PRM800K. Notably, our SPC is trained using only short CoT data, yet it can still generalize to the DeepSeek-R1-Distill-Qwen-7B model, which outputs in a long CoT style. It achieves 94.0% accuracy on MATH500, whereas Math-Shepherd and Qwen2.5-Math-7B-PRM800K achieve only 89.2% and 91.8%, respectively.

4.3 Ablation Study

The Impact of Different Strategies on Evolving Critic In Fig. 3 (left), we test critic models on ProcessBench, demonstrating the impact of different adversarial training methods. We refer to the sneaky generator and critic initialized after SFT as Sneaky-0 and Critic-0, respectively, while Sneaky-n and Critic-n represent models trained with n rounds of self-play adversarial data. In round 1, Sneaky-1 and Critic-1 are trained using data generated from the adversarial game between

Table 3: Performance of various methods for assisting different LLMs in math reasoning. By integrating Self-Consistency with our SPC, we achieve the best results across three types of LLMs on MATH500 and AIME2024 datasets.

Solvers	Verifiers	MATH500	AIME2024
Llama-3.1-8B-Instruct [10]	w/o	47.0	4.27
	Self-Consistency [2]	55.6	3.33
	Math-Shepherd [26]	52.4	3.33
	Qwen2.5-Math-7B-PRM800K [27]	54.6	3.33
	Self-Consistency + Math-Shepherd	53.6	6.67
	Self-Consistency + Qwen2.5-Math-7B-PRM800K	60.4	3.33
	SPC (Ours)	54.5	5.63
	Self-Consistency + SPC (Ours)	**62.8**	**6.67**
Qwen2.5-32B-Instruct [12]	w/o	78.0	14.4
	Self-Consistency	82.0	16.7
	Math-Shepherd	78.8	13.3
	Qwen2.5-Math-7B-PRM800K	82.8	16.7
	Self-Consistency + Math-Shepherd	80.8	13.3
	Self-Consistency + Qwen2.5-Math-7B-PRM800K	84.6	16.7
	SPC (Ours)	83.0	17.7
	Self-Consistency + SPC (Ours)	**85.2**	**23.3**
DeepSeek-R1-Distill-Qwen-7B [21]	w/o	87.7	53.8
	Self-Consistency	92.2	70.0
	Math-Shepherd	87.0	53.3
	Qwen2.5-Math-7B-PRM800K	84.2	63.3
	Self-Consistency + Math-Shepherd	89.2	60.0
	Self-Consistency + Qwen2.5-Math-7B-PRM800K	91.8	73.3
	SPC (Ours)	92.3	52.6
	Self-Consistency + SPC (Ours)	**94.0**	**73.3**

Sneaky-0 and Critic-0. For each successfully transformed erroneous step, we have the critic predict four critiques, which may include both correct and incorrect predictions, forming a pair of positive and negative samples with the same input but different outputs. **This method of constructing paired samples is more effective in RL training**, improving the critic from 70.8% in round 0 to 76.8%, whereas not constructing paired samples only achieves a performance of 75.0%.

For round 2, we explore two evolving strategies. (1) Generating round 2 data using the confrontation between Sneaky-1 and Critic-1 and mixing it with the data from round 1. We observe a significant performance decline in the critic trained with this setting, dropping from 76.8% to 72.0%, possibly due to overfitting. We notice that the win rate of Sneaky-1 against Critic-1 is only 13.2%. Therefore, such an overly unbalanced game might prevent the critic from learning new knowledge from the adversarial process, similar to how humans need opponents of comparable skill levels when playing chess. Therefore, we adopt another setting: (2) Generating data through the game between Sneaky-1 and Critic-0, given that Sneaky-1 had a win rate of 32.5% against Critic-0. We then mix the data from both rounds for training Critic-0 and update it as Critic-2. **Balancing the game prevents performance degradation and enables self-evolution**, improving SPC's performance to 77.7%.

The Performance of Sneaky Generator As shown in Fig. 3 (right), we analyze sneaky generators' success rates in attacking Qwen-2.5-7B-Instruct solver, as well as their win rates against Critic-0 and Critic-1. It is observed that the proportion of successful attacks on the solver gradually increases from 21.5% to 33.6%, as the sneaky generator iterates. We then feed successfully generated erroneous steps to the critic models. Sneaky generators' win rates against Critic-0 increase from 20.6% (Sneaky-0) to 30.3% (Sneaky-2). **Overall, the performance of the sneaky generators is iteratively improved.**

We also analyze a training setting without adding failed attacks on the solver as negative samples, using only successfully generated erroneous steps to construct positive/negative samples for training Sneaky-1, referred to as "w/o Reward from Solver" with lighter colors. We find that this approach severely impacts the performance of the sneaky generator, significantly reducing the proportion of successful attacks to 12.1%. Among the successfully attacked samples, the proportion that could deceive the critic is also very low, achieving a 19.6% win rate against Critic-0. Therefore, **it is crucial to ensure that the sneaky generator receives rewards from both the solver and the critic.**

5 Conclusion

In this paper, we propose a self-play critic with the ability of detecting step-level LLMs reasoning errors. Specifically, we design a sneaky generator to produce incorrect steps and a critic to assess the correctness of each step. Through the adversarial game between these two models, we can continuously generate positive and negative samples for reinforcement learning. The results on three reasoning process evaluation benchmarks fully demonstrate the effectiveness of our SPC. Furthermore, we apply SPC to assist LLMs' test-time search, further enhancing their reasoning performance.

References

[1] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[2] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[3] Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, et al. Thoughts are all over the place: On the underthinking of o1-like llms. *arXiv preprint arXiv:2501.18585*, 2025.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[5] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[6] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[7] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: A family of highly capable multimodal models, 2024.

[8] Anthropic. Introducing the next generation of claude, 2024. URL `https://www.anthropic.com/news/claude-3-family`.

[9] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[10] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[11] Qwen Team. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[12] Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL `https://qwenlm.github.io/blog/qwen2.5/`.

[13] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

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

[15] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

[16] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

[17] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[18] Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*, 2024.

[19] Jan Hendrik Kirchner, Yining Chen, Harri Edwards, Jan Leike, Nat McAleese, and Yuri Burda. Prover-verifier games improve legibility of llm outputs. *arXiv preprint arXiv:2407.13692*, 2024.

[20] OpenAI. Openai o1 system card. *preprint*, 2024.

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

[22] Qwen Team. Qwq: Reflect deeply on the boundaries of the unknown, November 2024. URL `https://qwenlm.github.io/blog/qwq-32b-preview/`.

[23] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=v8L0pN6EOi`.

[24] Skywork o1 Team. Skywork-o1 open series. `https://huggingface.co/Skywork`, November 2024. URL `https://huggingface.co/Skywork`.

[25] 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, 2022.

[26] Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2024.

[27] 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.

[28] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024. URL `https://openreview.net/forum?id=CxHRoTLmPX`.

[29] Yue Yu, Zhengxing Chen, Aston Zhang, Liang Tan, Chenguang Zhu, Richard Yuanzhe Pang, Yundi Qian, Xuewei Wang, Suchin Gururangan, Chao Zhang, et al. Self-generated critiques boost reward modeling for language models. *arXiv preprint arXiv:2411.16646*, 2024.

[30] Xin Zheng, Jie Lou, Boxi Cao, Xueru Wen, Yuqiu Ji, Hongyu Lin, Yaojie Lu, Xianpei Han, Debing Zhang, and Le Sun. Critic-cot: Boosting the reasoning abilities of large language model via chain-of-thoughts critic. *arXiv preprint arXiv:2408.16326*, 2024.

[31] Ruotian Ma, Peisong Wang, Cheng Liu, Xingyan Liu, Jiaqi Chen, Bang Zhang, Xin Zhou, Nan Du, and Jia Li. S^2r: Teaching llms to self-verify and self-correct via reinforcement learning. *arXiv preprint arXiv:2502.12853*, 2025.

[32] Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller llms stronger problem-solvers. *arXiv preprint arXiv:2408.06195*, 2024.

[33] Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Haitao Mi, and Dong Yu. Toward self-improvement of llms via imagination, searching, and criticizing. *arXiv preprint arXiv:2404.12253*, 2024.

[34] Zhen Huang, Haoyang Zou, Xuefeng Li, Yixiu Liu, Yuxiang Zheng, Ethan Chern, Shijie Xia, Yiwei Qin, Weizhe Yuan, and Pengfei Liu. O1 replication journey – part 2: Surpassing o1-preview through simple distillation big progress or bitter lesson? *Github*, 2024. URL `https://github.com/GAIR-NLP/O1-Journey`.

[35] Yancheng He, Shilong Li, Jiaheng Liu, Weixun Wang, Xingyuan Bu, Ge Zhang, Zhongyuan Peng, Zhaoxiang Zhang, Zhicheng Zheng, Wenbo Su, and Bo Zheng. Can large language models detect errors in long chain-of-thought reasoning?, 2025.

[36] Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023.

[37] AI-MO. Aime 2024, 2024. URL `https://huggingface.co/datasets/Maxwell-Jia/AIME_2024`.

[38] Meta. Introducing meta llama3: The most capable openly available llm to date, 2024. URL `https://ai.meta.com/blog/meta-llama-3/`.

[39] Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL `https://github.com/huggingface/open-r1`.

[40] Weihao Zeng, Yuzhen Huang, Wei Liu, Keqing He, Qian Liu, Zejun Ma, and Junxian He. 7b model and 8k examples: Emerging reasoning with reinforcement learning is both effective and efficient. `https://hkust-nlp.notion.site/simplerl-reason`, 2025. Notion Blog.

[41] Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. Tinyzero. https://github.com/Jiayi-Pan/TinyZero, 2025. Accessed: 2025-01-24.

[42] Ruihan Yang, Fanghua Ye, Jian Li, Siyu Yuan, Yikai Zhang, Zhaopeng Tu, Xiaolong Li, and Deqing Yang. The lighthouse of language: Enhancing llm agents via critique-guided improvement. *arXiv preprint arXiv:2503.16024*, 2025.

[43] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[44] Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[45] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[46] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[47] Pengyu Cheng, Tianhao Hu, Han Xu, Zhisong Zhang, Yong Dai, Lei Han, Xiaolong Li, et al. Self-playing adversarial language game enhances llm reasoning. *Advances in Neural Information Processing Systems*, 37:126515–126543, 2025.

[48] Ziyu Ye, Rishabh Agarwal, Tianqi Liu, Rishabh Joshi, Sarmishta Velury, Quoc V Le, Qijun Tan, and Yuan Liu. Evolving alignment via asymmetric self-play. *arXiv preprint arXiv:2411.00062*, 2024.

[49] Keming Lu, Bowen Yu, Chang Zhou, and Jingren Zhou. Large language models are superpositions of all characters: Attaining arbitrary role-play via self-alignment. *arXiv preprint arXiv:2401.12474*, 2024.

[50] Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. Self-play preference optimization for language model alignment. *arXiv preprint arXiv:2405.00675*, 2024.

[51] Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, et al. Justice or prejudice? quantifying biases in llm-as-a-judge. *arXiv preprint arXiv:2410.02736*, 2024.

[52] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

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

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

[55] Qwen. Qwen2.5-math-7b, 2024. URL `https://huggingface.co/Qwen/Qwen2.5-Math-7B`.

[56] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[57] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

[58] Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Sec. 1

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Sec. A

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: [NA]

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Sec. C

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: we upload the data and code.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Sec. C

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: we averaged the results with high randomness.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Sec. C

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: we have reviewed this.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Sec. A

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to

generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Sec. C

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: [NA]

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: [NA]

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: [NA]

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: Sec. 3

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Appendices

A Limitations and Societal Impact

Our SPC continuously generates data for reinforcement learning through adversarial games and has achieved impressive results. However, our current experiments are limited to several representative mathematical reasoning tasks. In future work, we plan to extend our approach to more general domains to further demonstrate the potential of the proposed framework.

Potential negative societal impacts of this work may include the misuse of a sneaky generator. For example, training a general sneaky generator to produce false and misleading information. On the other hand, enhancing the robustness of LLMs against attacks and training a general critic to automate the review of false information on the internet are also worthwhile research directions.

B Prompts

We demonstrate the prompts for generating data to initialize the sneaky generator and critic model.

Fig. 4 shows the prompts for querying GPT-4 (gpt-4-turbo-2024-04-09) to obtain sneaky transformations. These prompts include the five predefined error types, as well as correct/incorrect step pairs extracted from PRM800K. GPT-4 needs to first select a corresponding error type and then output the transformation process.

Figs. 5 and 6 illustrate the prompts we used to collect step-level critiques for initializing the critic model. We first use the prompts in Fig. 5 to feed data from PRM800K into DeepSeek-R1-Distill-Qwen-7B, collecting a batch of raw critiques. However, these critiques often do not follow our instructions in a standard format, making it difficult for us to assess the correctness of the critiques. Additionally, the responses are often too lengthy and include a lot of reflection and exploration, which is not conducive to performing SFT on the base model. Therefore, we feed these raw critiques (referred to as draft critiques in the prompt) into GPT-4o (gpt-4o-2024-08-06) for refinement, as shown in Fig. 6. By leveraging GPT-4o's strong instruction-following capabilities, we summarize the unstructured critiques into a concise and standardized version for subsequent SFT training.

When actually training the sneaky generator and critic model, we make slight modifications to the prompts mentioned above to avoid including incorrect steps that should be in the LLM output and unnecessary information, such as draft critiques. As shown in Figs. 7 and 8, we demonstrate the prompt templates for training the sneaky generator and critic model, respectively. These templates also remain unchanged during testing, data generation for self-play, and reinforcement learning processes.

C More Details

C.1 Evaluation Details

PRM800K [23] is a dataset collected by OpenAI for training and evaluating process supervision models. It is large in scale, containing 800K GPT-generated reasoning steps with human-annotated correctness. Additionally, PRM800K includes many pairs of correct and incorrect steps that share a common partial solution. Therefore, we construct 1,341 pairs of steps from the test split to evaluate model performance.

ProcessBench [27] is a benchmark with human-annotated step correctness, but it only includes a test set for evaluating models. Compared to PRM800k, the reasoning steps in ProcessBench are more diverse, comprising 3,400 cases from 12 different LLMs. All of these are math problems sourced from four datasets, including GSM8K [56], MATH [54], OlympiadBench [57] and Omni-MATH [58]. For incorrect solutions, we only retain the first incorrect step, while we randomly sample one correct step from a correct solution. We then feed the mixed 1,700 correct steps and 1,700 incorrect steps along with their corresponding partial solutions into the critic models.

DeltaBench [35] is the newest process benchmark focusing on evaluating long CoT collected from different open-source reasoning models, such as R1 [21] and QwQ [22]. We only utilize the math-related problems in this benchmark to evaluate model performance. Similarly, we retain the

labeled erroneous steps and sample the same number of correct steps, totaling 1,542. Given that existing PRMs baselines and our adversarial data generation process only collect short CoT data, this benchmark is more challenging and can be utilized to evaluate the effectiveness of our critic models in generalizing to popular reasoning models with long CoTs.

MATH500 [23, 54] and AIME2024 [37] are two highly popular benchmarks used to assess the mathematical reasoning abilities of LLMs. The former consists of 500 competition-level math problems, while the latter is derived from the American Invitational Mathematics Examination 2024. We evaluate the performance of LLMs on these two benchmarks when assisted by different verifiers in reasoning.

C.2 Preparing Data

For the SFT phase of the critic, we utilize the reasoning process data from PRM800K, along with prompting GPT-4 and DeepSeek-R1-Distill-Qwen-7B [21], to generate the required step-level critique data. We employ human annotations from PRM800 to filter out correctly generated data, ultimately obtaining 21.8K data, including 9.4K correct steps and 12.4K incorrect steps. As for the sneaky generator, we also prompt GPT-4 to teach the LLM to transform the correct steps from PRM800K into incorrect steps, finally collecting 13K data for SFT.

During the self-play phase, we use problems from the training set of PRM800K [23] to generate adversarial data for reinforcement learning. We use a total of three types of LLM solvers (Llama-3.1-8B-Instruct [10], Qwen2.5-7B-Instruct and Qwen2.5-32B-Instruct [12])) to provide the initial reasoning steps, in order to sample the correct steps and perform a sneaky transformation, which are then fed to the critic for adversarial self-play. Since we need the LLM to complete from both the original correct step and the sneaky step generated by the sneaky generator and compare the problem-solving success rate to determine whether the sneaky step contains an error that can truly affect the reasoning process, we first pre-generate 4 solutions for each problem and filter out those that have a success rate of 0, which are inherently unsolvable. For the remaining problems, we consider those with a success rate of 1/4 and 2/4 as medium difficulty level for this LLM, while those with a success rate of 3/4 and 4/4 are considered relatively easy. We primarily use medium-level problems to construct the training data, which ultimately accounts for 90% of the dataset, while easy problems are retained at about 10% because the model can already solve these problems smoothly without much additional learning and we only need a small amount of such data. These filtered medium-level problems will have 16 solutions generated by each LLM solver and easy problems will directly use the pregenerated 4 solutions. For each correctly predicted solution, one correct step is sampled and performed sneaky transformation. The successfully transformed incorrect steps are then further filtered for adversarial self-play.

The first round of self-play occurs between two SFT models (sneaky-0 and critic-0). We collect 6.4K data for the critic model for reinforcement learning, with a 1:1 ratio of positive to negative samples. Meanwhile, the sneaky generator receives 6K data, divided equally (2K each) into three scenarios: failing to attack the LLM solver, successfully attacking the LLM solver but losing to the critic, and successfully attacking the LLM solver while defeating the critic. The collected data in round 1 help us iteratively update the models to sneaky-1 and critic-1. As mentioned in Sec. 3.5, we balance the adversarial game to collect training data. Therefore, the second round of self-play occurs between sneaky-1 and critic-0. We further collect 6.8K data for the critic model, maintaining a 1:1 ratio between positive and negative samples, while continuing to gather 6K data for the sneaky generator, with the three scenarios still evenly distributed at 1/3 each. Finally, the data from two self-play rounds is merged to conduct offline reinforcement learning on sneaky-0 and critic-0, updating them to sneaky-2 and critic-2, respectively.

C.3 Training Hyperparameters

In the SFT initialization phase for both sneaky generator and critic models, we employ a batch size of 64 and a learning rate of 5e-6. We train the models for 3 epochs, with the maximum sequence length set to 4,096. To ensure both stability and convergence during training, we also incorporate a KL penalty into the training loss, setting the KL coefficient at 0.1. During the reinforcement learning of the self-play phase, we keep the batch size as 64 but use a learning rate of 2e-6. Except for setting

the KL coefficient at 0.1, we also add an SFT loss with a coefficient of 0.15 to ensure the stability of RL training.

D Case Analysis

As shown in Fig. 9, we present a comparison of critiques provided by two SPCs trained in round 0 and round 2. The input to the SPCs includes not only the system prompt (Sec. B but also the problem, partial solution, and the last step of the current reasoning process within the blue box in the figure. The last step contains a logical error, mistakenly identifying two expressions as unmatched. The critique from the round 0 SPC considers the last step to be correct, agreeing with the view that the expressions are inconsistent. However, the round 2 SPC, evolved through self-play training, accurately identifies the type of error, namely a logical error (underlined in the figure), recognizing that the two sides of the equation can be equivalently substituted. The entire analysis process is clear and coherent, ultimately leading to the correct prediction that this step is incorrect.

User Prompt

You are a math teacher. Given a math problem and a partial solution, you need to convert the last step of this partial solution to an incorrect step, while ensuring the incorrect step is subtle enough to be easily overlooked.

You should follow these steps to converse the last step:
1. Analyze the given partial solution and the last step. Clearly explain the solving process in the last step.
2. Choose an appropriate error type from the "Predefined Error Types" to complete the error generation. Specify your error generation method based on the current case, making the error less noticeable.
3. Step-by-step, write out the detailed error generation process for converting the Correct Last Step into the Incorrect Step.
4. Wrap the final incorrect step with <Answer> </Answer>.

Note: For each question, you will be given a reference incorrect last step. You need to convert to this reference incorrect last step, but **you must not reveal that you know this reference step in advance during the conversion process!**

Predefined Error Types
Error Type 1: Logical Error
Reference cases:
- incorrect orientation of geometric figures
- systematic counting error
- incomplete and inaccurate listing of factors
- Incorrect interpretation and connection of definitions
Error Type 2: calculation error
Reference cases:
- make incorrect simplification
- make incorrect factorization
- invalid algebraic operation
- substitution error
Error Type 3: Misunderstanding the Conditions.
Reference cases:
- Use imcomplete Condition
- Use contradictory condition
- misinterpretation of problem requirements
Error Type 4: Use Incorrect Rules/Formulas/Properties
Reference cases:
- incorrect identification of prime numbers
- misapplication of properties of complex numbers
- introduction of irrelevant inequality
Error Type 5: Incorrect Approach
Reference cases:
- incomplete solution

Your Task

Problem
{problem}

Partial Solution
{partial_solution}

Correct Next Step
{correct_step}

(Reference) Incorrect Next Step
{incorrect_step}

Your response

Figure 4: Prompt for querying GPT-4 to collect raw data of sneaky transformation CoT.

User Prompt

You are a helpful critic. Given a math Problem, a Partial Solution, the current Last Step of the solution, You need to provide a critique for the correctness of the Last Step.

You need to response a step-by-step analysis:
1. Analyzing the general thought of the Partial Solution.
2. Critique. You should write a brief critique here. This part should also maintain logical coherence with the summary of the general thought of the Partial Solution.
3. Conclusion. At the end of the response, output \\boxed{{Correct}} or \\boxed{{Incorrect}} to represent the correctness of the Last Step.

Problem
{problem}

Partial Solution
{partial_solution}

Last Step
{last_step}

Figure 5: Prompt for querying DeepSeek-R1-Distill-Qwen-7B to collect raw critiques with long CoT.

System Prompt

You are a helpful critic. Given a math Problem, a Partial Solution, the current Last Step of the solution, You need to provide a critique for the correctness of the Last Step. You already have a Draft Critique, so you only need to rewrite it in a clearer and more concise format.

You need to response a step-by-step analysis:
1. Analyzing the general thought of the Partial Solution. If the Draft Critique includes this analysis, you can directly summarize from it.
2. Critique. You should write a new version of brief critique here. The current Draft Critique is accurate but may contain redundant information, such as extensive consideration and attempts at derivation and analysis of the problem. You only need to select the useful analysis for the Last Step from it. This part should also maintain logical coherence with the summary of the general thought of the Partial Solution.
3. Conclusion. Please draw a conclusion about the correctness of the Last Step. Based on the analysis of the provided Draft Critique, determine whether the Last Step is correct or incorrect. If it is incorrect, you should also summarize a specific type of error, such as calculation error, logical error, etc. At the end of the response, output <Answer>Correct</Answer> or <Answer>Incorrect</Answer> to represent the correctness of the Last Step.

NOTE:
1. You need to refer to the draft critique, but pretend you didn't know this information beforehand, avoiding phrases like "the critique". Just write a new version of the critique for the Problem, Partial Solution, and Last Step.
2. In your revised version of Critique, you only need to focus on the Last Step, and it is not necessary to solve the problem to obtain the final answer.
3. DO NOT write the conclusion first and then the explanations for it. Instead, in the Critique, you start with an analysis of the Last Step. Then, in the Conclusion, drawing a conclusion about whether the Last Step is correct or incorrect.
4. The current partial solution may be incomplete, which is not an error type. You only need to focus on the correctness of the Last Step.

User Prompt

Problem
{problem}

Partial Solution
{partial_solution}

Last Step
{last_step}

Draft Critique
{draft_critique}

Figure 6: Prompt for querying GPT-4o to rewrite a long critique into a brief and standardized critique.

System Prompt

You are a math teacher. Given a math problem and a partial solution, you need to convert the last step of this partial solution to an incorrect step, while ensuring the incorrect step is subtle enough to be easily overlooked.

You should follow these steps to converse the last step:
1. Analyze the given partial solution and the last step. Clearly explain the solving process in the last step.
2. Choose an appropriate error type from the "Predefined Error Types" to complete the error generation. Specify your error generation method based on the current case, making the error less noticeable.
3. Step-by-step, write out the detailed error generation process for converting the Correct Last Step into the Incorrect Step.
4. Wrap the final incorrect step with <Answer> </Answer>.

Predefined Error Types
Error Type 1: Logical Error
Reference cases:
- incorrect orientation of geometric figures
- systematic counting error
- incomplete and inaccurate listing of factors
- Incorrect interpretation and connection of definitions
Error Type 2: calculation error
Reference cases:
- make incorrect simplification
- make incorrect factorization
- invalid algebraic operation
- substitution error
Error Type 3: Misunderstanding the Conditions.
Reference cases:
- Use Imcomplete Condition
- Use contradictory condition
- misinterpretation of problem requirements
Error Type 4: Use Incorrect Rules/Formulas/Properties
Reference cases:
- incorrect identification of prime numbers
- misapplication of properties of complex numbers
- introduction of irrelevant inequality
Error Type 5: Incorrect Approach
Reference cases:
- incomplete solution

User Prompt

Your Task

Problem
{problem}

Partial Solution
{partial_solution}

Correct Last Step
{correct_last_step}

Your response

Figure 7: Prompt for training the sneaky generator.

System Prompt

You are a helpful critic. Given a math Problem, a Partial Solution, the current Last Step of the solution, You need to provide a critique for the correctness of the Last Step.

You need to response a step-by-step analysis:
1. Analyzing the general thought of the Partial Solution.
2. Critique. You should write a brief critique here. This part should also maintain logical coherence with the summary of the general thought of the Partial Solution.
3. Conclusion. At the end of the response, output <Answer>Correct</Answer> or <Answer>Incorrect</Answer> to represent the correctness of the Last Step.

User Prompt

Problem
{problem}

Partial Solution
{partial_solution}

Last Step
{last_step}

Figure 8: Prompt for training the critic model.

SPC Input

Problem
Find all positive integers $n, k_1, \dots, k_n$ such that $k_1 + \cdots + k_n = 5n-4$ and \[\frac{1}{k_1} + \cdots + \frac{1}{k_n} = 1. \]

Partial Solution
To solve the problem, we need to find all positive integers \(n, k_1, \dots, k_n \) such that: First, \(k_1 + k_2 + \cdots + k_n = 5n - 4 \) and Second, \(\frac{1}{k_1} + \frac{1}{k_2} + \cdots + \frac{1}{k_n} = 1 \). We start by analyzing the second condition. The equation \(\frac{1}{k_1} + \frac{1}{k_2} + \cdots + \frac{1}{k_n} = 1 \) suggests that \(k_1, k_2, \ldots, k_n \) are the denominators of a unit fraction sum that equals 1. A well-known example is the Egyptian fraction representation.

Last Step
For small values of \(n \), we can try to find solutions manually. First, consider the case \(n = 1 \). If \(n = 1 \), then \(k_1 = 5(1) - 4 = 1 \). However, \(\frac{1}{k_1} = 1 \) does not satisfy \(k_1 = 1 \).

SPC Critique (Round 0)

Analysis of the General Thought of the Partial Solution
The partial solution begins by analyzing the given conditions: \(k_1 + k_2 + \cdots + k_n = 5n - 4 \) and \(\frac{1}{k_1} + \frac{1}{k_2} + \cdots + \frac{1}{k_n} = 1 \). It correctly identifies that the second condition suggests the use of Egyptian fractions, which are sums of distinct unit fractions that add up to 1. The solution then proceeds to test small values of \(n \) to find potential solutions.

Critique
The last step of the partial solution considers the case \(n = 1 \). It calculates \(k_1 = 5(1) - 4 = 1 \) and then checks if \(\frac{1}{k_1} = 1 \) satisfies the condition \(k_1 = 1 \). The solution correctly identifies that this does not satisfy the equation since \(k_1 \) must be 1, but the sum of the reciprocals would be 1, which is consistent but does not meet the requirement of distinct denominators for \(n = 1 \).

Conclusion
The correctness of the last step is <Answer>**Correct**</Answer>.

SPC Critique (Round 2)

Analysis of the General Thought of the Partial Solution
The partial solution begins by analyzing the given conditions: \(k_1 + k_2 + \cdots + k_n = 5n - 4 \) and \(\frac{1}{k_1} + \frac{1}{k_2} + \cdots + \frac{1}{k_n} = 1 \). It correctly identifies that the second condition suggests the use of Egyptian fractions, which are sums of distinct unit fractions. The solution then proceeds to test small values of \(n \) to find potential solutions.

Critique
In the last step, the solution considers the case \(n = 1 \). It correctly calculates \(k_1 = 5(1) - 4 = 1 \). However, it then incorrectly concludes that \(\frac{1}{k_1} = 1 \) does not satisfy \(k_1 = 1 \). This is **a logical error**. The equation \(\frac{1}{k_1} = 1 \) does indeed satisfy \(k_1 = 1 \), as substituting \(k_1 = 1 \) into the equation gives \(\frac{1}{1} = 1 \), which is true.

Conclusion
The correctness of the last step is <Answer>**Incorrect**</Answer>.

Figure 9: SPC critiques on ProcessBench before and after self-play training.