S^{2}R: Teaching LLMs to Self-verify and Self-correct via Reinforcement Learning

Ruotian Ma[1]*, Peisong Wang[2]*, Cheng Liu[1], Xingyan Liu[1], Jiaqi Chen[3], Bang Zhang[1], Xin Zhou[4], Nan Du[1]† , Jia Li [5]†

[1]Tencent [2]Tsinghua University
[3]The University of Hong Kong [4]Fudan University
[5]The Hong Kong University of Science and Technology (Guangzhou)

ruotianma@tencent.com, wps22@mails.tsinghua.edu.cn

Abstract

Recent studies have demonstrated the effectiveness of LLM test-time scaling. However, existing approaches to incentivize LLMs' deep thinking abilities generally require large-scale data or significant training efforts. Meanwhile, it remains unclear how to improve the thinking abilities of less powerful base models. In this work, we introduce S^{2}R, an efficient framework that enhances LLM reasoning by teaching models to self-verify and self-correct during inference. Specifically, we first initialize LLMs with iterative self-verification and self-correction behaviors through supervised fine-tuning on carefully curated data. The self-verification and self-correction skills are then further strengthened by both outcome-level and process-level reinforcement learning, with minimized resource requirements, enabling the model to adaptively refine its reasoning process during inference. Our results demonstrate that, with only 3.1k self-verifying and self-correcting behavior initialization samples, Qwen2.5-math-7B achieves an accuracy improvement from 51.0% to 81.6%, outperforming models trained on an equivalent amount of long-CoT distilled data. Extensive experiments and analysis based on three base models across both in-domain and out-of-domain benchmarks validate the effectiveness of S^{2}R. Our code and data are available at https://github.com/NineAbyss/S2R.

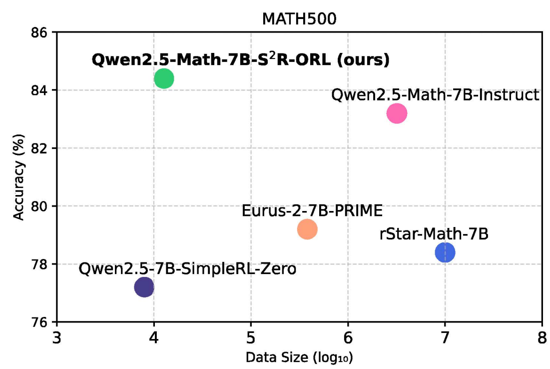

Figure 1: The data efficiency of S^{2}R compared to competitive methods, with all models initialized from Qwen2.5-Math-7B.

1 Introduction

Recent advancements in Large Language Models (LLMs) have demonstrated a paradigm shift from scaling up training-time efforts to test-time compute (Snell et al., 2024a; Kumar et al., 2024; Qi et al., 2024; Yang et al., 2024). The effectiveness of scaling test-time compute is illustrated by OpenAI o1 (OpenAI, 2024), which shows strong reasoning abilities by performing deep and thorough thinking, incorporating essential skills like self-checking, self-verifying, self-correcting and self-exploring during the model's reasoning process. This paradigm not only enhances reasoning in domains like mathematics and science but also offers new insights into improving the generalizability, helpfulness and safety of LLMs across various general tasks (OpenAI, 2024; Guo et al., 2025).

Recent studies have made various attempts to replicate the success of o1. These efforts include using large-scale Monte Carlo Tree Search (MCTS) to construct long-chain-of-thought (long-CoT) training data, or to scale test-time reasoning to improve the performance of current models (Guan et al., 2025; Zhao et al., 2024; Snell et al., 2024b); constructing high-quality long-CoT data for effective behavior cloning with substantial human effort (Qin et al., 2024); and exploring reinforcement learning to enhance LLM thinking abilities on large-scale training data and models (Guo et al., 2025; Team et al., 2025; Cui et al., 2025; Yuan et al., 2024). Recently, DeepSeek R1 (Guo et al., 2025) demonstrated that large-scale reinforcement learning can incentivize LLM's deep thinking abilities, with the R1 series showcasing

* Equal contribution. This work was done during Peisong, Cheng, Jiaqi and Bang were interning at Tencent.
† Corresponding authors.

the promising potential of long-thought reasoning. However, these approaches generally requires significant resources to enhance LLMs' thinking abilities, including large datasets, substantial training-time compute, and considerable human effort and time costs. Meanwhile, it remains unclear how to incentivize valid thinking in smaller or less powerful LLMs beyond distilling knowledge from more powerful models.

In this work, we propose S^2R, an efficient alternative to enhance the thinking abilities of LLMs, particularly for smaller or less powerful LLMs. Instead of having LLMs imitate the thinking process of larger, more powerful models, S^2R focus on teaching LLMs to think deeply by iteratively adopting two critical thinking skills: self-verifying and self-correcting. By acquiring these two capabilities, LLMs can continuously reassess their solutions, identify mistakes during solution exploration, and refine previous solutions after self-checking. Such a paradigm also enables flexible allocation of test-time compute to different levels of problems. Our results show that, with only 3.1k training samples, Qwen2.5-math-7B significantly benefits from learning self-verifying and self-correcting behaviors, achieving a 51.0% to 81.6% accuracy improvement on the Math500 test set. This performance outperforms the same base model distilled from an equivalent amount of long-CoT data (accuracy 80.2%) from QwQ-32B-Preview (Team, 2024a).

More importantly, S^2R employs both outcome-level and process-level reinforcement learning (RL) to further enhance the LLMs' self-verifying and self-correcting capabilities. Using only rule-based reward models, RL improves the validity of both the self-verification and self-correction process, allowing the models to perform more flexible and effective test-time scaling through a self-directed trial-and-error process. By comparing outcome-level and process-level RL for our task, we found that process-level supervision is particularly effective in boosting accuracy of the thinking skills at intermediate steps, which might benefit base models with limited reasoning abilities. In contrast, outcome-level supervision enables models explore more flexible trial-and-error paths towards the correct final answer, leading to consistent improvement in the reasoning abilities of more capable base models. Additionally, we further show the potential of offline reinforcement learning as a more efficient alternative to the online RL training.

We conducted extensive experiments across 3 LLMs on 7 math reasoning benchmarks. Experimental results demonstrate that S^2R outperforms competitive baselines in math reasoning, including recently-released advanced o1-like models Eurus-2-7B-PRIME (Cui et al., 2025), rStar-Math-7B (Guan et al., 2025) and Qwen2.5-7B-SimpleRL (Zeng et al., 2025). We also found that S^2R is generalizable to out-of-domain general tasks, such as MMLU-PRO, highlighting the validity of the learned self-verifying and self-correcting abilities. Additionally, we conducted a series of analytical experiments to better demonstrate the reasoning mechanisms of the obtained models, and provide insights into performing online and offline RL training for enhancing LLM reasoning.

2 Methodology

The main idea behind teaching LLMs self-verification and self-correction abilities is to streamline deep thinking into a critical paradigm: self-directed trial-and-error with self-verification and self-correction. Specifically: (1) LLMs are allowed to explore any potential (though possibly incorrect) solutions, especially when tackling difficult problems; (2) during the process, self-verification is essential for detecting mistakes on-the-fly; (3) self-correction enables the model to fix detected mistakes. This paradigm forms an effective test-time scaling approach that is more accessible for less powerful base models and is generalizable across various tasks.

In this section, we first formally define the problem (§2.1). Next, we present the two-stage training framework of S^2R, as described in Figure 2:

Stage 1: Behavior Initialization: We first construct dynamic self-verifying and self-correcting trial-and-error trajectories to initialize the desired behavior. Then, we apply supervised fine-tuning (SFT) to the initial policy models using these trajectories, resulting in behavior-initialized policy models (§2.2);

Stage 2: Reinforcement Learning: Following behavior initialization, we employ reinforcement learning to further enhance the self-verifying and self-correcting capabilities of the policy models. We explore both outcome-level and process-level RL methods, as well as their offline versions (§2.3).

2.1 Problem Setup

We formulate the desired LLM reasoning paradigm as a sequential decision-making process under a

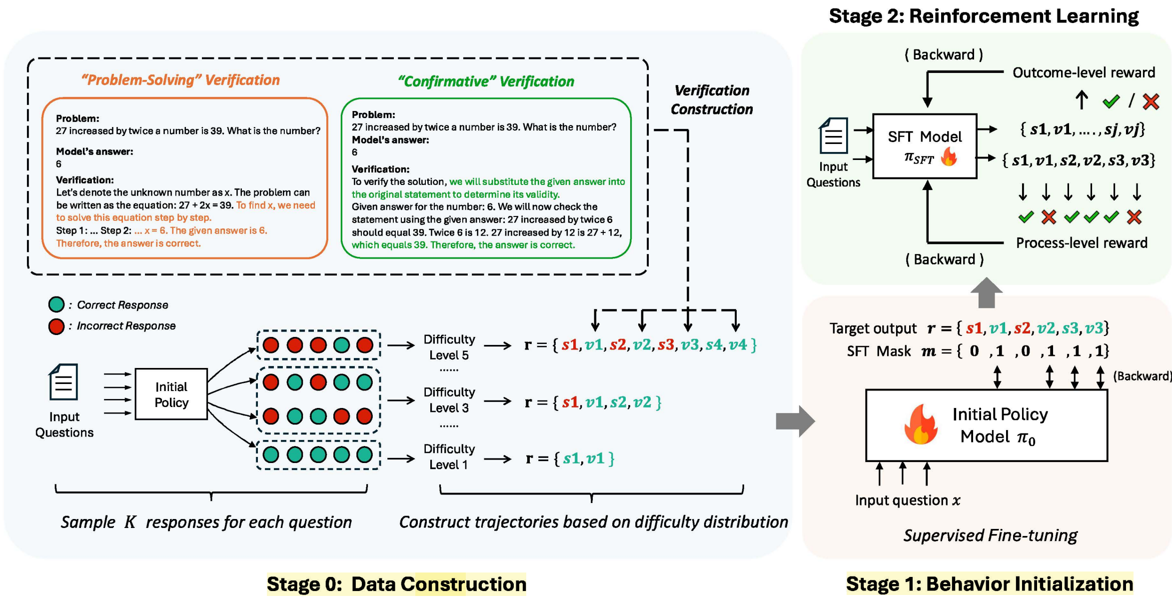

Figure 2: Overview of S^{2}R.

reinforcement learning framework. Given a problem x, the language model policy π is expected to generate a sequence of interleaved reasoning actions $y = (a_1, a_2, \cdots, a_T)$ until reaching the termination action <end>. We represent the series of actions before an action $a_t \in y$ as $y_{:a_t}$, i.e., $y_{:a_t} = (a_1, a_2, \cdots, a_{t-i})$, where a_t is excluded. The number of tokens in y is denoted as $|y|$, and the total number of actions in y is denoted as $|y|_a$.

We restrict the action space to three types: "`solve`", "`verify`", and "<end>", where "`solve`" actions represent direct attempts to solve the problem, "`verify`" actions correspond to self-assessments of the preceding solution, and "<end>" actions signal the completion of the reasoning process. We denote the type of action a_i as $Type(\cdot)$, where $Type(a_i) \in \{\texttt{verify}, \texttt{solve}, \texttt{<end>}\}$. We expect the policy to learn to explore new solutions by generating "`solve`" actions, to self-verify the correctness of preceding solutions with "`verify`" actions, and to correct the detected mistakes with new "`solve`" actions if necessary. Therefore, for each action a_i, the type of the next action a_{i+1} is determined by the following rules:

$$Type(a_{i+1}) = \begin{cases} \texttt{verify}, & Type(a_i) = \texttt{solve} \\ \texttt{solve}, & Type(a_i) = \texttt{verify} \\ & \text{ and Parser}(a_i) = \text{INCORRECT} \\ \texttt{<end>}, & Type(a_i) = \texttt{verify} \\ & \text{ and Parser}(a_i) = \text{CORRECT} \end{cases}$$

Here, $Parser(a) \in \{\text{CORRECT}, \text{INCORRECT}\}$ (for any action a where $Type(a) = \texttt{verify}$) is a function (e.g., a regex) that converts the model's free-form verification text into binary judgments.

For simplicity, we denote the j-th solve action as s_j and the j-th verify action as v_j. Then we have $y = (s_1, v_1, s_2, v_2, \cdots, s_k, v_k, \texttt{<end>})$.

2.2 Initializing Self-verification and Self-correction Behaviors

2.2.1 Learning Valid Self-verification

Learning to perform valid self-verification is the most crucial part in S^{2}R, as models can make mistakes during trial-and-error, and recognizing intermediate mistakes is critical for reaching the correct answer. In this work, we explore two methods for constructing self-verification behavior.

"Problem-Solving" Verification The most intuitive method for verification construction is to directly query existing models to generate verifications on the policy models' responses, and then filter for valid verifications. By querying existing models using different prompts, we found that existing models tend to perform verification in a "Problem-Solving" manner, i.e., by re-solving the problem and checking whether the answer matches the given one. We refer to this kind of verification as "Problem-Solving" Verification.

"Confirmative" Verification "Problem-solving" verification is intuitively not the ideal verification behavior we seek. Ideally, we expect the model to think outside the box and re-examine the solution from a new perspective, rather than thinking from the same problem-solving view for verification. We refer to this type of verification behavior as “Confirmative” Verification. Specifically, we construct “Confirmative” Verification by prompting existing LLMs to "verify the correctness of the answer without re-solving the problem", and filtering out invalid verifications using LLM-as-a-judge. The detail implementation can be found in Appendix §A.1.

2.2.2 Learning Self-correction

Another critical part of $\mathrm{S^2R}$ is enabling the model to learn self-correction. Inspired by Kumar et al. (2024) and Snell et al. (2024b), we initialize the self-correcting behavior by concatenating a series of incorrect solutions (each followed by a verification recognizing the mistakes) with a final correct solution. As demonstrated by Kumar et al. (2024), LLMs typically fail to learn valid self-correction behavior through SFT, but the validity of self-correction can be enhanced through reinforcement learning. Therefore, we only initialize the self-correcting behavior at this stage, leaving further enhancement of the self-correcting capabilities to the RL stage.

2.2.3 Constructing Dynamic Trial-and-Error Trajectory

We next construct the complete trial-and-error trajectories for behavior initialization SFT, following three principles:

- To ensure the diversity of the trajectories, we construct trajectories of various lengths. Specifically, we cover $k \in \{1, 2, 3, 4\}$ for $y = (a_1, \cdots, a_{2k}) = (s_1, v_1, \cdots, s_k, v_k)$ in the trajectories.

- To ensure that the LLMs learn to verify and correct their own errors, we construct the failed trials in each trajectory by sampling and filtering from the LLMs’ own responses.

- As a plausible test-time scaling method allocates reasonable effort to varying levels of problems, it is important to ensure the trial-and-error trajectories align with the difficulty level of problems. Specifically, more difficult problems will require more trial-and-error iterations before reaching the correct answer. Thus, we determine the length of each trajectory based on the accuracy of the sampled responses for each base model.

2.2.4 Supervised Fine-tuning for Thinking Behavior Initialization

Once the dynamic self-verifying and self-correcting training data $\mathcal{D}_{SFT}$ is ready, we optimize the policy π for thinking behavior initialization by minimizing the following objective:

$$\mathcal{L} = -\mathbb{E}_{(x,y)\sim\mathcal{D}_{SFT}} \sum_{a_t \in y} \delta_{mask}(a_t) \log \pi(a_t \mid x, y_{:a_t}) \tag{1}$$

where the mask function $\delta_{mask}(a_t)$ for action a_t in $y = (a_1, \cdots, a_T)$ is defined as:

$$\delta_{mask}(a_t) = \begin{cases} 1, & \text{if } Type(a_t) = \texttt{verify} \\ 1, & \text{if } Type(a_t) = \texttt{solve} \text{ and } t = T-1 \\ 1, & \text{if } Type(a_t) = \texttt{<end>} \text{ and } t = T \\ 0, & \text{otherwise} \end{cases}$$

That is, we optimize the probability of all verifications and only the last correct solution s_N by using masks during training.

2.3 Boosting Thinking Capabilities via Reinforcement Learning

After Stage 1, we initialized the policy model π with self-verification and self-correction behavior, obtaining π_{SFT}. We then explore further enhancing these thinking capabilities of π_{SFT} via reinforcement learning. Specifically, we explore two simple RL algorithms: the outcome-level REINFORCE Leave-One-Out (RLOO) algorithm and a proces-level group-based RL algorithm.

2.3.1 Outcome-level RLOO

We first introduce the outcome-level REINFORCE Leave-One-Out (RLOO) algorithm (Ahmadian et al., 2024; Kool et al., 2019) to further enhance the self-verification and self-correction capabilities of π_{SFT}. Given a problem x and the response $y = (s_1, v_1, ..., s_T, v_T)$, we define the reward function $R_o(x, y)$ based on the correctness of the last solution s_T:

$$R_o(x, y) = \begin{cases} 1, & V_{golden}(s_T) = \texttt{correct} \\ -1, & otherwise \end{cases}$$

Here $V_{golden}(\cdot) \in \{\texttt{correct}, \texttt{incorrect}\}$ represents ground-truth validation by matching the golden answer with the given solution. We calculate the advantage of each response y using an estimated baseline and KL reward shaping as follows:

$$A(x, y) = R_o(x, y) - \hat{b} - \beta \log \frac{\pi_{\theta_{old}}(y|x)}{\pi_{ref}(y|x)} \tag{2}$$

where β is the KL divergence regularization coefficient, and π_{ref} is the reference policy (in our case, π_{SFT}). $\hat{b}(x, y^{(m)}) = \frac{1}{M-1}\sum_{\substack{j=1,\ldots,M\\ j\neq m}} R_o(x, y^{(j)})$ is the baseline estimation of RLOO, which represents the leave-one-out mean of M sampled outputs $\{y^{(1)}, \ldots y^{(M)}\}$ for each input x, serving as a baseline estimation for each $y^{(m)}$. Then, we optimize the policy π_θ by minimizing the following objective after each sampling episode based on $\pi_{\theta_{old}}$:

$$\mathcal{L}(\theta) = -\mathbb{E}_{\substack{x\sim\mathcal{D}\\ y\sim\pi_{\theta_{\text{old}}}(\cdot|x)}}\Big[\min\big(r(\theta)A(x,y),\; \text{clip}\big(r(\theta), 1-\epsilon, 1+\epsilon\big)A(x,y)\big)\Big] \tag{3}$$

where $r(\theta) = \frac{\pi_\theta(y|x)}{\pi_{\theta_{\text{old}}}(y|x)}$ is the probability ratio.

When implementing the above loss function, we treat y as a complete trajectory sampled with an input problem x, meaning we optimize the entire trajectory with outcome-level supervision. With this approach, we aim to incentivize the policy model to explore more dynamic self-verification and self-correcting trajectories on its own, which has been demonstrated as an effective practice in recent work (Guo et al., 2025; Team et al., 2025).

2.3.2 Process-level Group-based RL

Process-level supervision has demonstrated effectiveness in math reasoning (Lightman et al., 2023a; Wang et al., 2024c). Since the trajectory of S²R thinking is naturally divided into self-verification and self-correction processes, it is intuitive to adopt process-level supervision for RL training.

Inspired by RLOO and process-level GRPO (Shao et al., 2024), we designed a group-based process-level optimization method. Specifically, we regard each action a in the output trajectory y as a sub-process and define the action level reward function $R_a(a \mid x, y_{:a})$ based on the action type. For each "`solve`" action s_j, we expect the policy to generate the correct solution; for each "`verify`" action v_j, we expect the verification to align with the actual solution validity. The corresponding rewards are defined as follows:

$$R_a(s_j \mid x, y_{:s_j}) = \begin{cases} 1, & V_{golden}(s_j) = \texttt{correct} \\ -1, & otherwise \end{cases}$$

$$R_a(v_j \mid x, y_{:v_j}) = \begin{cases} 1, & Parser(v_j) = V_{golden}(s_j) \\ -1, & otherwise \end{cases}$$

To calculate the advantage of each action a_t, we estimate the baseline as the average reward of the group of actions sharing the same **reward context**:

$$\mathbf{R}(a_t \mid x, y) = (R_a(a_i \mid x, y_{:a_i}))_{i=1}^{t-1}$$

which is defined as the reward sequence of the previous actions $y_{:a_t}$ of each action a_t. We denote the set of actions sharing the same reward context $\mathbf{R}(a_t \mid x, y)$ as $\mathcal{G}(\mathbf{R}(a_t \mid x, y))$. Then the baseline can be estimated as follows:

$$\hat{b}(a_t \mid x, y) = \frac{1}{|\mathcal{G}(\mathbf{R}(a_t|x,y))|}\sum_{a\in\mathcal{G}(\mathbf{R}(a_t|x,y))} R_a(a|x^{(a)}, y_{:a}^{(a)}) \tag{4}$$

And the advantage of each action a_t is:

$$A(a_t \mid x, y) = R_a(a_t \mid x, y_{:a_t}) - \hat{b}(a_t \mid x, y) - \beta \log\frac{\pi_{\theta_{old}}(a_t \mid x, y)}{\pi_{\text{ref}}(a_t \mid x, y)} \tag{5}$$

The main idea of the group-based baseline estimation is that the actions sharing the same reward context are provided with similar amounts of information before the action is taken. For instance, all actions sharing a reward context consisting of one failed attempt and one successful verification (i.e., $\mathbf{R}(a_t|x,y) = (-1, 1)$) are provided with the information about the problem, a failed attempt, and the reassessment on the failure. Given the same amount of information, it is reasonable to estimate a baseline using the average reward of these actions.

Putting it all together, we minimize the following surrogate loss function to update the policy parameters θ, using trajectories collected from π_{old}:

$$\mathcal{L}(\theta) = -\mathbb{E}_{\substack{x\sim\mathcal{D}\\ y\sim\pi_{\theta_{\text{old}}}(\cdot|x)}}\Big[\frac{1}{|y|_a}\sum_{a\in y}\min\big(r_a(\theta)A(a|x,y_{:a}),\; \text{clip}\big(r_a(\theta), 1-\epsilon, 1+\epsilon\big)A(a|x,y_{:a})\big)\Big] \tag{6}$$

where $r_a(\theta) = \frac{\pi_\theta(a|x,y_{:a})}{\pi_{\theta_{\text{old}}}(a|x,y_{:a})}$ is the importance ratio.

2.4 More Efficient Training with Offline RL

While online RL is known for its high resource requirements, offline RL, which does not require real-time sampling during training, offers a more efficient alternative for RL training. Additionally, offline sampling allows for more accurate baseline

Stage 1: Behavior Initialization		
Base Model	**Source**	**# Training Data**
Llama-3.1-8B-Instruct	MATH	4614
Qwen2-7B-Instruct	MATH	4366
Qwen2.5-Math-7B	MATH	3111
Stage 2: Reinforcement Learning		
Base Model	**Source**	**# Training Data**
Llama-3.1-8B-Instruct	MATH+GSM8K	9601
Qwen2-7B-Instruct	MATH+GSM8K	9601
Qwen2.5-Math-7B	MATH+OpenMath2.0	10000

Table 1: Training data statistics.

calculations with better trajectories grouping for each policy. As part of our exploration into more efficient RL training in S²R framework, we also experimented with offline RL to assess its potential in further enhancing the models' thinking abilities. In Appendix §D.2, we include more details and formal definition for offline RL training.

3 Experiment

To verify the effectiveness of the proposed method, we conducted extensive experiments across 3 different base policy models on various benchmarks.

3.1 Experiment Setup

Base Models To evaluate the general applicability of our method across different LLMs, we conducted experiments using three distinct base models: Llama-3.1-8B-Instruct (Dubey et al., 2024), Qwen2-7B-Instruct (qwe, 2024), and Qwen2.5-Math-7B (Qwen, 2024). Llama-3.1-8B-Instruct and Qwen2-7B-Instruct are versatile general-purpose models trained on diverse domains without a specialized focus on mathematical reasoning. In contrast, Qwen2.5-Math-7B is a state-of-the-art model specifically tailored for mathematical problem-solving and has been widely adopted in recent research on math reasoning (Guan et al., 2025; Cui et al., 2025; Zeng et al., 2025).

Training Data Setup For *Stage 1: Behavior Initialization*, we used the widely adopted MATH (Hendrycks et al., 2021a) training set for dynamic trial-and-error data collection [1]. For each base model, we sampled 5 responses per problem in the training data. After data filtering and sampling, we constructed a dynamic trial-and-error training set consisting of 3k-4k instances for each base model. Detailed statistics of the training set are shown in Table 1. For *Stage 2: Reinforcement Learning*, we used the MATH+GSM8K (Cobbe et al., 2021a) training data for RL training on the policy π_{SFT} initialized from Llama-3.1-8B-Instruct and Qwen2-7B-Instruct. Since Qwen2.5-math-7b already achieves high accuracy on the GSM8K training data after Stage 1, we additionally include training data randomly sampled from the OpenMath2 dataset (Toshniwal et al., 2024). Following (Cui et al., 2025), we filter out excessively easy or difficult problems based on each π_{SFT} from Stage 1 to enhance the efficiency and stability of RL training, resulting in RL training sets consisting of approximately 10000 instances. Detailed statistics of the final training data can be found in Table 1. Additional details on training data construction can be found in in Appendix §A.1.

[1]We use the MATH split from Lightman et al. (2023a), i.e., 12000 problems for training and 500 problems for testing.

Baselines We benchmark our proposed method against four categories of strong baselines:

- ***Frontier LLMs*** includes cutting-edge proprietary models such as GPT-4o, the latest Claude, and OpenAI's o1-preview and o1-mini.
- ***Top-tier open-source reasoning models*** covers state-of-the-art open-source models known for their strong reasoning capabilities, including Mathstral-7B-v0.1 (Team, 2024b), NuminaMath-72B (LI et al., 2024), LLaMA3.1-70B-Instruct (Dubey et al., 2024), and Qwen2.5-Math-72B-Instruct (Yang et al., 2024).
- ***Enhanced models built on Qwen2.5-Math-7B***: Given the recent popularity of Qwen2.5-Math-7B as a base policy model, we evaluate S²R against three competitive baselines that have demonstrated superior performance based on Qwen2.5-Math-7B: Eurus-2-7B-PRIME (Cui et al., 2025), rStar-Math-7B (Guan et al., 2025), and Qwen2.5-7B-SimpleRL (Zeng et al., 2025). These models serve as direct and strong baseline for our Qwen2.5-Math-7B-based variants.
- ***SFT with different CoT constructions***: We also compare with training on competitive types of CoT reasoning, including the original CoT solution in the training datasets, and Long-CoT solutions distilled from QwQ-32B-Preview (Team, 2024a), a widely adopted open-source o1-like model (Chen et al., 2024c; Guan et al., 2025; Zheng et al., 2024). Specifically, to ensure a fair comparison between behavior initialization with long-CoT and S²R, we use long-CoT data of the same size as our behavior initialization data. We

Model	Datasets							Average
	MATH 500	AIME 2024	AMC 2023	College Math	Olympiad Bench	GSM8K	GaokaoEn 2023	
Frontier LLMs								
GPT-4o*	76.6	9.3	47.5	48.5	43.3	92.9	67.5	55.1
Claude3.5-Sonnet*	78.3	16.0	-	-	-	96.4	-	-
GPT-o1-preview*	85.5	44.6	90.0	-	-	-	-	-
GPT-o1-mini*	90.0	56.7	95.0	57.8	65.3	94.8	78.4	76.9
Top-tier Open-source Reasoning LLMs								
Mathstral-7B-v0.1*	57.8	0.0	37.5	33.7	21.5	84.9	46.0	40.2
NuminaMath-72B-CoT*	64.0	3.3	70.0	39.7	32.6	90.8	58.4	51.3
LLaMA3.1-70B-Instruct*	65.4	23.3	50.0	42.5	27.7	94.1	54.0	51.0
Qwen2.5-Math-72B-Instruct*	85.6	30.0	70.0	49.5	49.0	95.9	71.9	64.6
General Model: Llama-3.1-8B-Instruct								
Llama-3.1-8B-Instruct	48.0	6.7	30.0	30.8	15.6	84.4	41.0	36.6
Llama-3.1-8B-Instruct + Original Solution SFT	31.0	3.3	7.5	22.0	8.0	58.7	28.3	22.7
Llama-3.1-8B-Instruct + Long CoT SFT	51.4	6.7	27.5	36.3	19.0	87.0	**48.3**	39.5
Llama-3.1-8B-S^{2}R-BI (*ours*)	49.6	**10.0**	20.0	33.3	17.6	85.3	41.0	36.7
Llama-3.1-8B-S^{2}R-PRL (*ours*)	53.6	6.7	25.0	33.7	18.5	86.7	43.1	38.2
Llama-3.1-8B-S^{2}R-ORL (*ours*)	**55.0**	6.7	**32.5**	**34.7**	**20.7**	**87.3**	45.2	**40.3**
General Model: Qwen2-7B-Instruct								
Qwen2-7B-Instruct	51.2	3.3	30.0	18.2	19.1	86.4	39.0	35.3
Qwen2-7B-Instruct + Original Solution SFT	41.2	0.0	25.0	30.1	10.2	74.5	34.8	30.8
Qwen2-7B-Instruct + Long CoT SFT	60.4	6.7	32.5	36.3	23.4	81.2	53.5	42.0
Qwen2-7B-S^{2}R-BI (*ours*)	61.2	3.3	27.5	**41.1**	**27.1**	87.4	49.1	42.4
Qwen2-7B-S^{2}R-PRL (*ours*)	**65.4**	6.7	35.0	36.7	27.0	**89.0**	49.9	**44.2**
Qwen2-7B-S^{2}R-ORL (*ours*)	64.8	3.3	**42.5**	34.7	26.2	86.4	**50.9**	44.1
Math-Specialized Model: Qwen2.5-Math-7B								
Qwen2.5-Math-7B	51.0	16.7	45.0	21.5	16.7	58.3	39.7	35.6
Qwen2.5-Math-7B-Instruct	83.2	13.3	72.5	47.0	40.4	**95.6**	67.5	59.9
Eurus-2-7B-PRIME*(Cui et al., 2025)	79.2	26.7	57.8	45.0	42.1	88.0	57.1	56.6
rStar-Math-7B*[2](Guan et al., 2025)	78.4	26.7	47.5	**52.5**	**47.1**	89.7	65.7	58.2
Qwen2.5-7B-SimpleRL*(Zeng et al., 2025)	82.4	26.7	62.5	-	43.3	-	-	-
Qwen2.5-Math-7B + Original Solution SFT	58.0	6.7	42.5	35.8	20.0	79.5	51.9	42.1
Qwen2.5-Math-7B + Long CoT SFT	80.2	16.7	60.0	49.6	42.1	91.4	69.1	58.4
Qwen2.5-Math-7B-S^{2}R-BI (*ours*)	81.6	23.3	60.0	43.9	44.4	91.9	70.1	59.3
Qwen2.5-Math-7B-S^{2}R-PRL (*ours*)	83.4	26.7	70.0	43.8	46.4	93.2	**70.4**	62.0
Qwen2.5-Math-7B-S^{2}R-ORL (*ours*)	**84.4**	23.3	**77.5**	43.8	44.9	92.9	70.1	**62.4**

Table 2: The performance of S^{2}R and other strong baselines on the most challenging math benchmarks is presented. **BI** refers to the behavior-initialized models through supervised fine-tuning, **ORL** denotes models trained with outcome-level RL, and **PRL** refers to models trained with process-level RL. The highest results are highlighted in **bold** and the second-best results are marked with underline. For some baselines, we use the results from their original reports or from Guan et al. (2025), denoted by *.

provide more details on the baseline data construction in Appendix §A.2.3.

More details on the baselines are included in Appendix §A.2.

Evaluation Datasets We evaluate the proposed method on 7 diverse mathematical benchmarks. To ensure a comprehensive evaluation, in addition to the in-distribution GSM8K (Cobbe et al., 2021b) and MATH500 (Lightman et al., 2023a) test sets, we include challenging out-of-distribution benchmarks covering various difficulty levels and mathematical domains, including the AIME 2024 competition problems (AI-MO, 2024a), the AMC 2023 exam (AI-MO, 2024b), the advanced reasoning tasks from Olympiad Bench (He et al., 2024), and college-level problem sets from College Math (Tang et al., 2024a). Additionally, we assess performance on real-world standardized tests, the GaoKao (Chinese College Entrance Exam) En 2023 (Liao et al., 2024). A detailed description of these datasets is provided in Appendix §B.1.

Evaluation Metrics We report Pass@1 accuracy for all baselines. For inference, we employ vLLM (Kwon et al., 2023) and develop evaluation scripts based on Qwen Math's codebase. All evaluations are performed using greedy decoding. Details of the prompts used during inference are provided in

[2] To ensure a fair comparison, we report the Pass@1 (greedy) accuracy obtained without the process preference model of rStar, rather than the result obtained with increased test-time computation using 64 trajectories.

Appendix §A.3. All implementation details, including hyperparameter settings, can be found in Appendix §B.2.

3.2 Main Results

Table 2 shows the main results of S^{2}R compared with baseline methods. We can observe that: (1) S^{2}R consistently improves the reasoning abilities of models across all base models. Notably, on Qwen2.5-Math-7B, the proposed method improves the base model by 32.2% on MATH500 and by 34.3% on GSM8K. (2) Generally, S^{2}R outperforms the baseline methods derived from the same base models across most benchmarks. Specifically, on Qwen2.5-Math-7B, S^{2}R surpasses several recently proposed competitive baselines, such as Eurus-2-7B-PRIME, rStar-Math-7B and Qwen2.5-7B-SimpleRL. While Eurus-2-7B-PRIME and rStar-Math-7B rely on larger training datasets (Figure 1) and require more data construction and reward modeling efforts, S^{2}R only needs linear sampling efforts for data construction, 10k RL training data and rule-based reward modeling. These results highlight the efficiency of S^{2}R. (3) With the same scale of SFT data, S^{2}R also outperforms the long-CoT models distilled from QwQ-32B-Preview, demonstrating that learning to self-verify and self-correct is an effective alternative to long-CoT for test-time scaling in smaller LLMs.

Comparing process-level and outcome-level RL, we find that outcome-level RL generally outperforms process-level RL across the three models. This is likely because outcome-level RL allows models to explore trajectories without emphasizing intermediate accuracy, which may benefit enhancing long-thought reasoning in stronger base models like Qwen2.5-Math-7B. In contrast, process-level RL, which provides guidance for each intermediate verification and correction step, may be effective for models with lower initial capabilities, such as Qwen2-7B-Instruct. As shown in Figure 3, process-level RL can notably enhance the verification and correction abilities of Qwen2-7B-S^{2}R-BI.

3.3 Generalizing to Cross-domain Tasks

Despite training on math reasoning tasks, we found that the learned self-verifying and self-correcting capability can also generalize to out-of-distribution general domains. In Table 3, we evaluate the SFT model and the outcome-level RL model based on Qwen2.5-Math-7B on four cross-domain tasks: FOLIO (Han et al., 2022) on logical reasoning, CRUXEval (Gu et al., 2024) on code reasoning, StrategyQA (Geva et al., 2021) on multi-hop reasoning and MMLUPro-STEM on multi-task complex understanding (Wang et al., 2024d; Shen et al., 2025), with details of these datasets provided in Appendix §B.1. The results show that after learning to self-verify and self-correct, the proposed method effectively boosts the base model's performance across all tasks and achieves comparative results to the baseline models. These findings indicate that the learned self-verifying and self-correcting capabilities are general thinking skills, which can also benefit reasoning in general domains. Additionally, we expect that the performance in specific domains can be further improved by applying S^{2}R training on domain data with minimal reward model requirements (e.g., rule-based or LLM-as-a-judge). For better illustration, we show cases on how the trained models perform self-verifying and self-correcting on general tasks in Appendix §E.

Model	FOLIO	CRUX-Eval	Strategy-QA	MMLUPro-STEM
Qwen2.5-Math-72B-Instruct	69.5	68.6	94.3	66.0
Llama-3.1-70B-Instruct*	65.0	59.6	88.8	61.7
OpenMath2-Llama3.1-70B*	68.5	35.1	95.6	55.0
QwQ-32B-Preview*	84.2	65.2	88.2	71.9
Eurus-2-7B-PRIME	56.7	<u>50.0</u>	79.0	**53.7**
Qwen2.5-Math-7B-Instruct	**61.6**	28.0	81.2	44.7
Qwen2.5-Math-7B	37.9	40.8	61.1	46.0
Qwen2.5-Math-7B-S^{2}R-BI (*ours*)	<u>58.1</u>	48.0	<u>88.7</u>	49.8
Qwen2.5-Math-7B-S^{2}R-ORL (*ours*)	**61.6**	**50.9**	**90.8**	<u>50.0</u>

Table 3: Performance of the proposed method and the baseline methods on 4 cross-domain tasks. The results with * are reported by Shen et al. (2025).

3.4 Analyzing Self-verification and Self-correction Abilities

In this section, we conduct analytical experiments on the models' self-verification and self-correction capabilities from various perspectives.

3.4.1 Problem-solving v.s. Confirmative Verification

We first compare the Problem-solving and Confirmative Verification methods described in §2.2.1. In Table 4, we present the verification results of different methods on the Math500 test set. We report the overall verification accuracy, as well as the initial verification accuracy when the initial answer is correct ($V_{golden}(s_0) = \texttt{correct}$) and incorrect ($V_{golden}(s_0) = \texttt{incorrect}$), respectively.

We observe from the table that: (1) Generally, problem-solving verification achieves superior overall accuracy compared to confirmative verification. This result is intuitive, as existing models

Base Model	Methods	Overall Verification Acc.	Initial Verification Acc.	
			$V_{golden}(s_0)$ = correct	$V_{golden}(s_0)$ = incorrect
Llama3.1-8B-Instruct	Problem-solving	80.10	87.28	66.96
	Confirmative	65.67	77.27	78.22
Qwen2-7B-Instruct	Problem-solving	73.28	90.24	67.37
	Confirmative	58.31	76.16	70.05
Qwen2.5-Math-7B	Problem-solving	77.25	91.21	56.67
	Confirmative	61.58	82.80	68.04

Table 4: Comparison of problem-solving and confirmative verification.

are trained for problem-solving, and recent studies have highlighted the difficulty of existing LLMs in performing reverse thinking (Berglund et al., 2023; Chen et al., 2024b). During data collection, we also found that existing models tend to verify through problem-solving, even when prompted to verify without re-solving (see Table 6 in Appendix §A.1). (2) In practice, accuracy alone does not fully reflect the validity of a method. For example, when answer accuracy is sufficiently high, predicting all answers as correct will naturally lead to high verification accuracy, but this is not a desired behavior. By further examining the initial verification accuracy for both correct and incorrect answers, we found that problem-solving verification exhibits a notable bias toward predicting answers as correct, while the predictions from confirmative verification are more balanced. We deduce that this bias arises might be because problem-solving verification is more heavily influenced by the preceding solution, aligning with previous studies showing that LLMs struggle to identify their own errors (Huang et al., 2023; Tyen et al., 2023). In contrast, confirmative verification performs verification from different perspectives, making it less influenced by the LLMs' preceding solution.

In all experiments, we used confirmative verification for behavior initialization.

3.4.2 Boosting Self-verifying and Self-correcting with RL

In this experiment, we investigate the effect of RL training on the models' self-verifying and self-correcting capabilities.

We assess self-verification using the following metrics: (1) ***Verification Accuracy***: The overall accuracy of verification predictions, as described in §3.4.1. (2) ***Error Recall***: The recall of verification when the preceding answers are incorrect. (3) ***Correct Precision***: The precision of verification when it predicts the answers as correct. Both Error

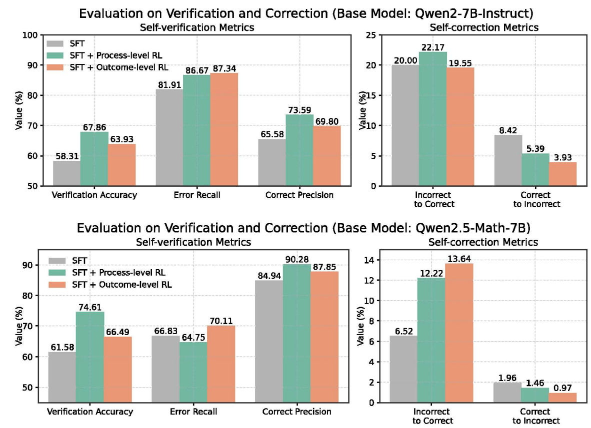

Figure 3: Evaluation on verification and correction.

Recall and Correct Precision directly affect the final answer accuracy: if verification fails to detect an incorrect answer, or if it incorrectly predicts an answer as correct, the final answer will be wrong.

For self-correction, we use the following metrics: (1) ***Incorrect to Correct Rate***: the rate at which the model successfully corrects an incorrect initial answer to a correct final answer. (2) ***Correct to Incorrect Rate***: the rate at which the model incorrectly changes a correct initial answer to an incorrect final answer. We provide the formal definitions of the metrics used in Appendix §C.

In Figure 3, we present the results of the behavior-initialized model (SFT) and different RL models obtained from Qwen2.5-Math-7B. We observe that: (1) Both RL methods effectively enhance self-verification accuracy. The process-level RL shows larger improvement on accuracy, while the outcome-level RL consistently improves Error Recall and Correct Precision. This might be because process-level supervision indiscriminately promotes verification accuracy in intermediate steps, while outcome-level supervision allows the policy model to explore freely in intermediate steps and only boosts the final answer accuracy, thus mainly enhancing Error Recall and Correct Precision (which directly relate to final answer accuracy). (2) Both RL methods can successfully enhance the models' self-correction capability. Notably, the model's ability to correct incorrect answers is significantly improved after RL training. The rate of model mistakenly altering correct answers is also notably reduced. This comparison demonstrates that S^{2}R can substantially enhance the validity of models' self-correction ability.

Model	MATH 500	AIME 2024	AMC 2023	College Math	Olympiad Bench	GSM8K	GaokaoEn 2023	Average
	Datasets							
General Model: Qwen2-7B-Instruct								
Qwen2-7B-Instruct	51.2	3.3	30.0	18.2	19.1	86.4	39.0	35.3
Qwen2-7B-S^{2}R-BI (*ours*)	61.2	3.3	27.5	**41.1**	**27.1**	87.4	49.1	42.4
Qwen2-7B-S^{2}R-PRL (*ours*)	**65.4**	6.7	35.0	36.7	27.0	**89.0**	49.9	44.2
Qwen2-7B-S^{2}R-ORL (*ours*)	64.8	3.3	**42.5**	34.7	26.2	86.4	**50.9**	44.1
Qwen2-7B–Instruct-S^{2}R-PRL-offline (*ours*)	61.6	**10.0**	32.5	40.2	26.5	87.6	50.4	44.1
Qwen2-7B-Instruct-S^{2}R-ORL-offline (*ours*)	61.0	6.7	37.5	40.5	27.3	87.4	49.6	**44.3**
Math-Specialized Model: Qwen2.5-Math-7B								
Qwen2.5-Math-7B	51.0	16.7	45.0	21.5	16.7	58.3	39.7	35.6
Qwen2.5-Math-7B-S^{2}R-BI (*ours*)	81.6	23.3	60.0	43.9	44.4	91.9	70.1	59.3
Qwen2.5-Math-7B-S^{2}R-PRL (*ours*)	83.4	**26.7**	70.0	43.8	46.4	**93.2**	70.4	62.0
Qwen2.5-Math-7B-S^{2}R-ORL (*ours*)	**84.4**	23.3	**77.5**	43.8	44.9	92.9	70.1	**62.4**
Qwen2.5-Math-7B-S^{2}R-PRL-offline (*ours*)	83.4	23.3	62.5	**50.0**	**46.7**	92.9	**72.2**	61.6
Qwen2.5-Math-7B-S^{2}R-ORL-offline (*ours*)	82.0	20.0	67.5	49.8	45.8	92.6	70.4	61.2

Table 5: Comparison of S^{2}R using online and offline RL training.

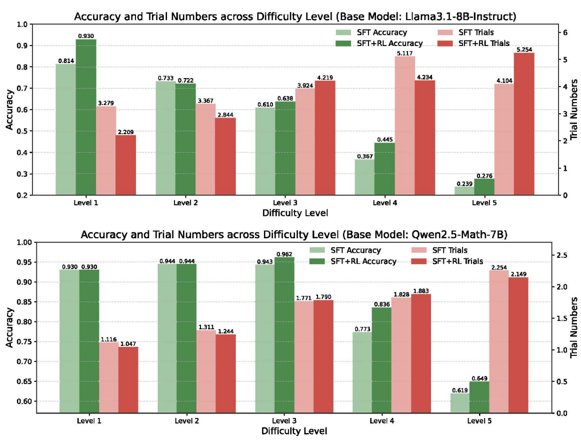

Figure 4: The accuracy and average trial number of different models across difficulty levels. Evaluated on MATH500 test set.

3.4.3 Improvement across Difficulty Levels

To further illustrate the effect of S^{2}R training, Figure 4 shows the answer accuracy and average number of trials (i.e., the average value of "K" across all $y = (s_1, v_1, \cdots, s_K, v_K)$ under each difficulty level) for the SFT and SFT+RL models. We observe that: (1) By learning to self-verify and self-correct during reasoning, the models learn to dynamically allocate test-time effort. For easier problems, the models can reach a confident answer with fewer trials, while for more difficult problems, they require more trials to achieve a confident answer. (2) RL further improves test-time effort allocation, particularly for less capable model (e.g., Llama3.1-8B-Instruct). (3) After RL training, the answer accuracy for more difficult problems is notably improved, demonstrating the effectiveness of the self-verifying and self-correcting paradigm in enhancing the models' reasoning abilities.

3.5 Exploring Offline RL

As described in §2.4, we explore offline RL as a more efficient alternative to online RL training, given the effectiveness of offline RL has been demonstrated in recent studies (Baheti et al., 2023; Cheng et al., 2025; Wang et al., 2024b).

Table 5 presents the results of offline RL with process-level and outcome-level supervision, compared to online RL. We can observe that: (1) Different from online RL, process-level supervision outperforms outcome-level supervision in offline RL training. This interesting phenomenon may be due to: a) Outcome-level RL, which excels at allowing models to freely explore dynamic trajectories, is more suitable for on-the-fly sampling during online parameter updating. b) In contrast, process-level RL, which requires accurate baseline estimation for intermediate steps, benefits from offline trajectory sampling, which can provide more accurate baseline estimates with larger scale data sampling. (2) Offline RL consistently improves performance over the behavior-initialized models across most benchmarks and achieves comparable results to online RL. These results highlight the potential of offline RL as a more efficient alternative for enhancing LLMs' deep reasoning.

4 Related Work

4.1 Scaling Test-time Compute

Scaling test-time compute recently garners wide attention in LLM reasoning (Snell et al., 2024b; Wu et al., 2024; Brown et al., 2024). Existing studies have explored various methods for scaling up test-time compute, including: (1) ***Aggregation-***

based methods that samples multiple responses for each question and obtains the final answer with self-consistency (Wang et al., 2023) or by selecting best-of-N answer using a verifier or reward model (Wang et al., 2024c; Zhang et al., 2024b; Lightman et al., 2023b; Havrilla et al., 2024b); (2) ***Search-based methods*** that apply search algorithms such as Monte Carlo Tree Search (Tian et al., 2024; Wang et al., 2024a; Zhang et al., 2024a; Qi et al., 2024), beam search (Snell et al., 2024b), or other effective algorithms (Feng et al., 2023; Yao et al., 2023) to search for correct trajectories; (3) ***Iterative-refine-based methods*** that iteratively improve test performance through self-refinement (Madaan et al., 2024a; Shinn et al., 2024; Chen et al., 2024a, 2025). Recently, there has been a growing focus on training LLMs to perform test-time search on their own, typically by conducting longer and deeper thinking (OpenAI, 2024; Guo et al., 2025). These test-time scaling efforts not only directly benefit LLM reasoning, but can also be integrated back into training time, enabling iterative improvement for LLM reasoning (Qin et al., 2024; Feng et al., 2023; Snell et al., 2024b; Luong et al., 2024). In this work, we also present an efficient framework for training LLMs to perform effective test-time scaling through self-verification and self-correction iterations. This approach is achieved without extensive efforts, and the performance of $\mathrm{S^2R}$ can also be consistently promoted via iterative training.

4.2 Self-verification and Self-correction

Enabling LLMs to perform effective self-verification and self-correction is a promising solution for achieving robust reasoning for LLMs (Madaan et al., 2024b; Shinn et al., 2023; Paul et al., 2023; Lightman et al., 2023a), and these abilities are also critical for performing deep reasoning. Previous studies have shown that direct prompting of LLMs for self-verification or self-correction is suboptimal in most scenarios (Huang et al., 2023; Tyen et al., 2023; Ma et al., 2024; Zhang et al., 2024c). As a result, recent studies have explored various approaches to enhance these capabilities during post-training (Saunders et al., 2022; Rosset et al., 2024; Kumar et al., 2024). These methods highlight the potential of using human-annotated or LLM-generated data to equip LLMs with self-verification or self-correction capabilities (Zhang et al., 2024d; Jiang et al., 2024), while also indicating that behavior imitation via supervised fine-tuning alone is insufficient for achieving valid self-verification or self-correction (Kumar et al., 2024; Qu et al., 2025; Kamoi et al., 2024). In this work, we propose effective methods to enhance LLMs' self-verification and self-correction abilities through principled imitation data construction and RL training, and demonstrate the effectiveness of our approach with in-depth analysis.

4.3 RL for LLM Reasoning

Reinforcement learning has proven effective in enhancing LLM performance across various tasks (Ziegler et al., 2019; Stiennon et al., 2020; Bai et al., 2022; Ouyang et al., 2022; Setlur et al., 2025). In LLM reasoning, previous studies typically employ RL in an actor-critic framework (Lightman et al., 2024; Tajwar et al., 2024; Havrilla et al., 2024a), and research on developing accurate reward models for RL training has been a long-standing focus, particularly in reward modeling for Process-level RL (Lightman et al., 2024; Setlur et al., 2024, 2025; Luo et al., 2024). Recently, several studies have demonstrate that simplified reward modeling and advantage estimation (Ahmadian et al., 2024; Shao et al., 2024; Team et al., 2025; Guo et al., 2025) in RL training can also effectively enhance LLM reasoning. Recent advances in improving LLMs' deep thinking (Guo et al., 2025; Team et al., 2025) further highlight the effectiveness of utilizing unhackable rewards (Gao et al., 2023; Everitt et al., 2021) to consistently enhance LLM reasoning. In this work, we also show that simplified advantage estimation and RL framework enable effective improvements on LLM reasoning. Additionally, we conducted an analysis on process-level RL, outcome-level RL and offline RL, providing insights for future work in RL for LLM reasoning.

5 Conclusion

In this work, we propose $\mathrm{S^2R}$, an efficient framework for enhancing LLM reasoning by teaching LLMs to iteratively self-verify and self-correct during reasoning. We introduce a principled approach for behavior initialization, and explore both outcome-level and process-level RL to further strengthen the models' thinking abilities. Experimental results across three different base models on seven math reasoning benchmarks demonstrate that $\mathrm{S^2R}$ significantly enhances LLM reasoning with minimal resource requirements. Since self-verification and self-correction are two crucial abilities for LLMs' deep reasoning, S^{2}R offers an interpretable framework for understanding how SFT and RL enhance LLMs' deep reasoning. It also offers insights into the selection of RL strategies for enhancing LLMs' long-CoT reasoning.

References

2024. Qwen2 technical report.

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

AI-MO. 2024a. Aime 2024.

AI-MO. 2024b. Amc 2023.

Ashutosh Baheti, Ximing Lu, Faeze Brahman, Ronan Le Bras, Maarten Sap, and Mark Riedl. 2023. Leftover lunch: Advantage-based offline reinforcement learning for language models. *arXiv preprint arXiv:2305.14718*.

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

Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. 2023. The reversal curse: Llms trained on" a is b" fail to learn" b is a". *arXiv preprint arXiv:2309.12288*.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Jiefeng Chen, Jie Ren, Xinyun Chen, Chengrun Yang, Ruoxi Sun, and Sercan Ö Arık. 2025. Sets: Leveraging self-verification and self-correction for improved test-time scaling. *arXiv preprint arXiv:2501.19306*.

Justin Chih-Yao Chen, Archiki Prasad, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. 2024a. Magicore: Multi-agent, iterative, coarse-to-fine refinement for reasoning. *arXiv preprint arXiv:2409.12147*.

Justin Chih-Yao Chen, Zifeng Wang, Hamid Palangi, Rujun Han, Sayna Ebrahimi, Long Le, Vincent Perot, Swaroop Mishra, Mohit Bansal, Chen-Yu Lee, et al. 2024b. Reverse thinking makes llms stronger reasoners. *arXiv preprint arXiv:2411.19865*.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. 2024c. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021a. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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

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

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

Tom Everitt, Marcus Hutter, Ramana Kumar, and Victoria Krakovna. 2021. Reward tampering problems and solutions in reinforcement learning: A causal influence diagram perspective. *Synthese*, 198(Suppl 27):6435–6467.

Xidong Feng, Ziyu Wan, Muning Wen, Ying Wen, Weinan Zhang, and Jun Wang. 2023. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*.

Leo Gao, John Schulman, and Jacob Hilton. 2023. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pages 10835–10866. PMLR.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies. *Transactions of the Association for Computational Linguistics (TACL)*.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. 2024. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*.

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*.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, et al. 2022. Folio: Natural language reasoning with first-order logic. *arXiv preprint arXiv:2209.00840*.

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. 2024a. Teaching large language models to reason with reinforcement learning. *arXiv preprint arXiv:2403.04642*.

Alex Havrilla, Sharath Raparthy, Christoforus Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, and Roberta Raileanu. 2024b. Glore: When, where, and how to improve llm reasoning via global and local refinements. *arXiv preprint arXiv:2402.10963*.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021a. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

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

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Huchen Jiang, Yangyang Ma, Chaofan Ding, Kexin Luan, and Xinhan Di. 2024. Towards intrinsic self-correction enhancement in monte carlo tree search boosted reasoning via iterative preference learning. *arXiv preprint arXiv:2412.17397*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Wouter Kool, Herke van Hoof, and Max Welling. 2019. Buy 4 reinforce samples, get a baseline for free!

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. 2024. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. 2024. Numinamath. `[https://github.com/project-numina/aimo-progress-prize](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf)`.

Minpeng Liao, Wei Luo, Chengxi Li, Jing Wu, and Kai Fan. 2024. Mario: Math reasoning with code interpreter output–a reproducible pipeline. *arXiv preprint arXiv:2401.08190*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023a. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023b. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

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

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

Trung Quoc Luong, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. Reft: Reasoning with reinforced fine-tuning. *arXiv preprint arXiv:2401.08967*.

Ruotian Ma, Xiaolei Wang, Xin Zhou, Jian Li, Nan Du, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Are large language models good prompt optimizers? *arXiv preprint arXiv:2402.02101*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024a. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024b. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

OpenAI. 2024. Openai o1 system card. *preprint*.

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

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2023. Refiner: Reasoning feedback on intermediate representations. *arXiv preprint arXiv:2304.01904*.

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

Yiwei Qin, Xuefeng Li, Haoyang Zou, Yixiu Liu, Shijie Xia, Zhen Huang, Yixin Ye, Weizhe Yuan, Hector Liu, Yuanzhi Li, et al. 2024. O1 replication journey: A strategic progress report–part 1. *arXiv preprint arXiv:2410.18982*.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. 2025. Recursive introspection: Teaching language model agents how to self-improve. *Advances in Neural Information Processing Systems*, 37:55249–55285.

Qwen. 2024. Qwen2.5-math-7b.

Corby Rosset, Ching-An Cheng, Arindam Mitra, Michael Santacroce, Ahmed Awadallah, and Tengyang Xie. 2024. Direct nash optimization: Teaching language models to self-improve with general preferences. *arXiv preprint arXiv:2404.03715*.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. 2022. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. 2025. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *Advances in Neural Information Processing Systems*, 37:43000–43031.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2024. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*.

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*.

Maohao Shen, Guangtao Zeng, Zhenting Qi, Zhang-Wei Hong, Zhenfang Chen, Wei Lu, Gregory Wornell, Subhro Das, David Cox, and Chuang Gan. 2025. Satori: Reinforcement learning with chain-of-action-thought enhances llm reasoning via autoregressive search. *arXiv preprint arXiv:2502.02508*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*.

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

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

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021.

Fahim Tajwar, Anikait Singh, Archit Sharma, Rafael Rafailov, Jeff Schneider, Tengyang Xie, Stefano Ermon, Chelsea Finn, and Aviral Kumar. 2024. Preference fine-tuning of llms should leverage suboptimal, on-policy data. *arXiv preprint arXiv:2404.14367*.

Zhengyang Tang, Xingxing Zhang, Benyou Wan, and Furu Wei. 2024a. Mathscale: Scaling instruction tuning for mathematical reasoning. *arXiv preprint arXiv:2403.02884*.

Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. 2024b. Mathscale: Scaling instruction tuning for mathematical reasoning. *arXiv preprint arXiv:2403.02884*.

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

Qwen Team. 2024a. Qwq: Reflect deeply on the boundaries of the unknown.

The Mistral AI Team. 2024b. Mathstral-7b-v0.1.

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

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. 2024. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. *arXiv preprint arXiv:2410.01560*.

Gladys Tyen, Hassan Mansoor, Peter Chen, Tony Mak, and Victor Cărbune. 2023. Llms cannot find reasoning errors, but can correct them! *arXiv preprint arXiv:2311.08516*.

Chaojie Wang, Yanchen Deng, Zhiyi Lv, Shuicheng Yan, and An Bo. 2024a. Q*: Improving multi-step reasoning for llms with deliberative planning. *Preprint*, arXiv:2406.14283.

Huaijie Wang, Shibo Hao, Hanze Dong, Shenao Zhang, Yilin Bao, Ziran Yang, and Yi Wu. 2024b. Offline reinforcement learning for llm multi-step reasoning. *arXiv preprint arXiv:2412.16145*.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. 2024c. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *Preprint*, arXiv:2312.08935.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. 2024d. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. 2024. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

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

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

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

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2024b. Generative verifiers: Reward modeling as next-token prediction. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*.

Qingjie Zhang, Han Qiu, Di Wang, Haoting Qian, Yiming Li, Tianwei Zhang, and Minlie Huang. 2024c. Understanding the dark side of llms' intrinsic self-correction. *arXiv preprint arXiv:2412.14959*.

Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. 2024d. Small language models need strong verifiers to self-correct reasoning. *arXiv preprint arXiv:2404.17140*.

Yu Zhao, Huifeng Yin, Bo Zeng, Hao Wang, Tianqi Shi, Chenyang Lyu, Longyue Wang, Weihua Luo, and Kaifu Zhang. 2024. Marco-o1: Towards open reasoning models for open-ended solutions. *arXiv preprint arXiv:2411.14405*.

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*.

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

A Implementation Details

A.1 Verification Processing and SFT Data Construction

Given the responses sampled from the original LLM policy, we prompt frontier LLMs for initial verifications. In order to construct more valid verification, we force the LLMs to "verify without re-solving the problem" and filter out invalid verifications during data processing. We found that despite being instructed to "verify without re-solving the problem", most existing LLMs still biased to solve the problem again, as shown in Table 6. Finally, we collected the verification data by querying gpt-4-preview-1106[1] , which shows strong instruction-following ability to "verify without re-solving the problem" and can perform plausible verification such as adopting reverse thinking, inductive reasoning and other methods.

For these collected prompts, we refine the remaining verifications using gpt-4o to improve fluency and clarity. During this refinement, we instruct gpt-4o to append a conclusion at the end of each verification based on its stance—for example: "Therefore, the answer is correct/incorrect/cannot verify." Finally, we discard any verifications where the judgment does not align with the actual correctness of the answer. The prompts we used during the whole process are provided in Appendix §A.3.

With the refined and filtered verifications, we construct the SFT data as follows. For each problem, we determine the number of answer attempts required to eventually obtain a correct answer based on the accuracy from the initial sampling. The lower the accuracy, the more rounds of responses are generated. In our implementation, we categorize all problems into four difficulty levels and construct answer sequences with 1, 2, 3, or 4 rounds, according to descending accuracy. Then, after an incorrect answer, we append "Wait, let me recheck my solution" along with the corresponding verification. If that answer is not the final attempt, we further append "Let me try again." We ensure that the last answer in the sequence is correct. Additionally, we ensure that the answers in each round for a given problem are distinct. Figure 5 is an example of SFT data constructed with 4 rounds of responses.

[1] `https://openai.com/api/`

A.2 Baseline Details

A.2.1 Baseline Implementations

In Table 2, the reported results for Frontier LLMs and Top-tier Open-source Reasoning LLMs are sourced from the original reports and Guan et al. (2025). We evaluate Llama-3.1-8B-Instruct (Dubey et al., 2024), Qwen2-7B-Instruct (qwe, 2024), Qwen2.5-Math-7B, Qwen2.5-Math-7B-Instruct and Qwen2.5-Math-72B-Instruct(Yang et al., 2024) using the same process described in Section §3.1. For Eurus-7B-PRIME (Cui et al., 2025), rStar-Math-7B (Guan et al., 2025), and Qwen2.5-7B-SimpleRL (Zeng et al., 2025), we report results directly from the original papers.

In Table 3, the results for Llama-3.1-70B-Instruct and QwQ-32B-Preview are taken from Shen et al. (2025). For the remaining baselines, we follow the official evaluation protocol of the dataset project[2].

A.2.2 Baseline License

In this work, we utilize the Llama-3.1-8B-Instruct model, whose license can be reviewed at `https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct/blob/main/LICENSE`. In addition, the models Qwen2-7B-Instruct, Qwen2.5-Math-7B, Eurus-2-7B-PRIME, and project vLLM are distributed under the Apache License 2.0. We gratefully acknowledge the contributions of the open-source community and strictly adhere to the terms of the respective licenses.

A.2.3 Baseline SFT Data Construction

Original Solution SFT Data In this setting, we use the solution from the original dataset as sft data. To ensure a fair comparison, we maintain the same training data volume as our behavior initialization approaches.

Long CoT SFT Data We also introduce a baseline by fine-tuning on Long CoT responses generated by QwQ-32B-Preview (Team, 2024a). Specifically, we instruct QwQ to generate responses to given problems and filter out those with incorrect answers. The remaining high-quality responses are then used for supervised fine-tuning. Importantly, we ensure that the total training data volume

[2] https://github.com/Yale-LILY/FOLIO
https://github.com/facebookresearch/cruxeval
https://github.com/eladsegal/strategyqa
https://github.com/TIGER-AI-Lab/MMLU-Pro

Without Asking for Confirmative Verification	
Model	Confirmative out of 100
GPT-4o	26
GPT-4-Preview-1106	32
QwQ-32B-preview	37
Llama-3.1-70B-Instruct	28
Asking for Confirmative Verification	
Model	Confirmative out of 100
GPT-4o	44
GPT-4-Preview-1106	**61**
QwQ-32B-preview	58
Llama-3.1-70B-Instruct	50

Table 6

remains consistent with that used in our behavior initialization approach. The prompt we use for QwQ is provided in Appendix §A.3.

A.3 Prompts

The prompts we use in all experiments are as follows:

Sampling Responses During Training/Inference

Please reason step by step, and put your final answer within \boxed{}.
Problem: {problem}

Verification Refinement

You are a math teacher. I will give you a math problem and an answer.
Verify the answer's correctness without step–by–step solving. Use alternative verification methods.
Question: {problem}
Answer: {answer}
Verification:

Verification Collection

Refine this verification text to read as a natural self–check within a solution. Maintain logical flow and professionalism.
Key Requirements:
1. Avoid phrases like "without solving step–by–step" or "as a math teacher".
2. Treat the answer as your own prior solution.
3. Conclude with EXACTLY one of:
Therefore, the answer is correct.
Therefore, the answer is incorrect.
Therefore, the answer cannot be verified.
Original text: {verification}

B Detailed Experiment Settings

B.1 Datasets

Details of each test dataset we used as benchmark are as follows:

B.1.1 In-domain Datasets

MATH500 (Lightman et al., 2023b) offers a streamlined slice of the broader MATH (Hendrycks et al., 2021b) dataset, comprising 500 test problems selected through uniform sampling. Despite its smaller scope, it maintains a distribution of topics and difficulty levels that mirrors the larger MATH corpus.

GSM8K (Cobbe et al., 2021a) features around 8,500 grade-school math word problems. The dataset focuses on simple arithmetic through early algebra and includes 1,319 distinct tasks in its test set.

OlympiadBench (He et al., 2024) collects 8,476 advanced math and physics questions drawn from Olympiad contexts, with some originating from the Chinese college entrance exam. We use the subset of 674 text-only competition questions, providing open-ended math challenges.

AMC2023 (AI-MO, 2024b) and **AIME** (AI-MO, 2024a) each supply a set of challenging exam-style problems: 40 questions from AMC 2023 and 30 from AIME 2024, all in text-only format.

CollegeMath (Tang et al., 2024b) is a dataset targeting advanced college-level mathematics, drawn from nine textbooks spanning seven major fields—algebra, pre-calculus, calculus, vector calculus, probability, linear algebra, and differential equations. The final collection comprises 1,281 training examples and 2,818 test examples.

Gaokao2023en (Liao et al., 2024) is a dataset consisting of 385 mathematics problems sourced from the 2023 Chinese higher education entrance examination, which have been professionally translated into English.

B.1.2 Cross-domain Datasets

FOLIO (Han et al., 2022) is meticulously annotated to assess intricate logical reasoning in natural language. It pairs 1,430 conclusions with 487 sets of premises—each verified using first-order logic (FOL)—and contains 203 unique problems in its test portion.

CRUXEval (Gu et al., 2024) tests code comprehension and reasoning through 800 concise Python functions (spanning 3–13 lines). Each function is accompanied by one or more input-output examples. The goal is to predict the correct outputs given the function body and a specific input. The test partition encompasses all 800 problems.

StrategyQA (Geva et al., 2021) targets multi-hop reasoning questions where the necessary inter-

Model	Learning Rate	Batch Size	KL Coefficient	Max Length	Training Epochs
Llama-3.1-8B-Instruct	5e-6	32	0.1	8000	3
Qwen2-7B-Instruct	5e-6	32	0.1	6000	3
Qwen2.5-Math-7B	5e-6	32	0.01	8000	3

Table 7: Model Training Hyperparameter Settings (SFT)

Model	Learning Rate	Training Batch Size	Forward Batch Size	KL Coefficient	Max Length	Sampling Temperature	Clip Range	Training Steps
Llama-3.1	5e-7	64	256	0.05	8000	0.7	0.2	500
Qwen2-7B-Instruct	5e-7	64	256	0.05	6000	0.7	0.2	500
Qwen2.5-Math-7B	5e-7	64	256	0.01	8000	0.7	0.2	500

Table 8: Model Training Hyperparameter Settings (RL)

mediate steps are not explicit. Each of its 2,780 items includes a strategic query, a breakdown of the reasoning steps, and supporting evidence drawn from Wikipedia.

MMLUProSTEM is extracted from **MMLU-Pro** (Wang et al., 2024d). Following Satori (Shen et al., 2025), we conduct evaluations on six STEM subsets—physics, chemistry, computer science, engineering, biology, and economics.

B.2 Hyperparameters Setting

During behavior initialization with SFT, we use a batch size of *32* and adopt a learning rate of *5e-6*. We set the maximum sequence length *8000* to accommodate long responses and verifications. To balance stability and convergence during training, we add a KL punishment to the training loss, and the KL coefficient is set to *0.1*.

During reinforcement learning, for each training batch, we use a training batch size of 64, and sample n responses for each question in a batch, resulting a forward batch size of $64n$. For each forward batch, we update the model for n step with the training batch size 64. Specifically, for both process-level and outcome-level RL, we adopt $n = 4$ (i.e., for RLOO, the sample number is also 4). More hyperparameters of the RL training are presented in Table 8. We use the BF16 model precision in all experiments.

Main hyperparameters used in the experiments are illustrated in Table 7 and 8.

B.3 Experiment Environment

All experiments are implemented using the PyTorch framework on 32 NVIDIA H20 (96GB) GPUs or 32 NVIDIA A100Pro (40GB) GPUs. Our training code is built upon Hugging Face TRL[3]. For inference, we use a single NVIDIA A100 (40GB) GPU with vLLM-0.5.4[4]. We utilize transformers version 4.39.3 for fine-tuning Qwen2-7B-Instruct and Qwen2.5-Math-7B, version 4.44.0 for fine-tuning Llama-3.1-8B, and version 4.46.3 for reinforcement learning. We use PyTorch 2.1.1 across our training pipeline. Our evaluation code is built upon Qwen Math's evaluation codebase[5].

C Metrics Definition

We include the formal definition of metrics we use for analyzing self-verification and self-correction behaviors of the post-trained models as follows.

C.1 Notations

We first present the main notations used in our formulation in Table 9.

C.2 Self-Verification Metrics

C.2.1 Verification Accuracy (VA)

Verification Accuracy measures how often the verification prediction matches the ground-truth correctness (N is the total number of verifications in the responses to the test set):

$$\text{VA} = \frac{1}{N} \sum_{t=1}^{N} \mathbb{I}\Big(\text{Parser}(v_t) = V_{golden}(s_t)\Big). \tag{7}$$

C.2.2 Error Recall (ER)

Error Recall measures the recall of detecting incorrect answers (i.e., the fraction of actually incorrect

[3] https://github.com/huggingface/trl
[4] https://github.com/vllm-project/vllm
[5] https://github.com/QwenLM/Qwen2.5-Math

Variable	Description
π	The policy
x	Problem instance
y	Series of predefined actions: $y = \{a_1, a_2, \ldots, a_n\}$
a_i	The i-th action in the response y, and let $Type(a_i) \in \{\texttt{verify}, \texttt{solve}, \texttt{<end>}\}$
s_j	j^{th} attempt to solve the problem
v_j	j^{th} self-verification for the j^{th} attempt
$Parser(\cdot)$	$Parser(v_j) \in \{\texttt{correct}, \texttt{incorrect}\}$ The text parser to get the self-verification result indicating the correctness of action s_j
$V_{golden}(\cdot)$	$V_{golden}(a_i) \in \{\texttt{correct}, \texttt{incorrect}\}$
$R(\cdot)$	The rule based reward function $R(\cdot) \in \{-1, 1\}$ $R(s_j) = \begin{cases} 1, & V_{golden}(s_j) = \texttt{correct} \\ -1, & otherwise \end{cases}$ $R(v_j) = \begin{cases} 1, & Parser(v_j) = V_{golden}(s_j) \\ -1, & otherwise \end{cases}$
<end>	End of action series
$\mathbb{I}(\cdot)$	The indicator function, $\mathbb{I}(\cdot) \in \{0, 1\}$. $\mathbb{I}(\cdot) = 1$ if the condition inside holds true, and $\mathbb{I}(\cdot) = 0$ otherwise.

Table 9: Variable Lookup Table

answers that are successfully identified as incorrect):

$$\text{ER} = \frac{\sum_y \sum_{t=1}^{\frac{|y|_a}{2}} \mathbb{I}\Big(R(s_t){=}{-1}\Big)\, \mathbb{I}\Big(\text{Parser}(v_t){=}\texttt{incorrect}\Big)}{\sum_y \sum_{t=1}^{\frac{|y|_a}{2}} \mathbb{I}\Big(R(s_t){=}{-1}\Big)}. \tag{8}$$

where $|y|_a$ is the total number of actions in y and $\frac{|y|_a}{2}$ is the total number of attempts to solve the problem ($y = \{a_1, a_2, \cdots, a_{|y|_a}\} = \{s_1, v_1, \cdots, s_{\frac{|y|_a}{2}}, v_{\frac{|y|_a}{2}}\}$).

C.2.3 Correct Precision (CP)

Correct Precision measures the precision when the verification model predicts an answer to be correct (i.e., among all "correct" predictions, how many are truly correct):

$$\text{CP} = \frac{\sum_y \sum_{t=1}^{\frac{|y|_a}{2}} \mathbb{I}\Big(\text{Parser}(v_t){=}\texttt{correct}\Big)\, \mathbb{I}\Big(R(s_t){=}1\Big)}{\sum_y \sum_{t=1}^{\frac{|y|_a}{2}} \mathbb{I}\Big(\text{Parser}(v_t){=}\texttt{correct}\Big)}. \tag{9}$$

C.3 Self-Correction Metrics

C.3.1 Incorrect to Correct Rate (ICR)

The rate at which the model successfully corrects an initially incorrect answer ($R(s_1) = -1$) into a correct final answer ($R(s_{T_y}) = 1$), where $T_y = |y|_a/2$ is the total number of attempts to solve the problem in each y. Formally:

$$\text{ICR} = \frac{\sum_y \mathbb{I}\big(R(s_1) = -1\big)\, \mathbb{I}\big(R(s_{T_y}) = 1\big)}{\sum_y \mathbb{I}\big(R(s_1) = -1\big)}. \tag{10}$$

C.3.2 Correct to Incorrect Rate (CIR)

The rate at which the model incorrectly alters an initially correct answer ($R(s_1) = 1$) into an incorrect final answer ($R(s_{T_y}) = -1$), where $T_y = |y|_a/2$ is the total number of attempts to solve the problem in each y. Formally:

$$\text{CIR} = \frac{\sum_y \mathbb{I}\big(R(s_1) = 1\big)\, \mathbb{I}\big(R(s_{T_y}) = -1\big)}{\sum_y \mathbb{I}\big(R(s_1) = 1\big)}. \tag{11}$$

D Offline RL Training Details

In this section, we provide additional details on the offline reinforcement learning training process, including formal definition, ablation studies, and implementation details.

D.1 Accuracy-Grouped Baseline Definition

To fully leverage the advantages of offline RL, which does not require real-time sampling, we explore more appropriate baseline selection by further grouping trajectories based on problem difficulty. Intuitively, for two trajectories $y^{(1)}$ and $y^{(2)}$ sampled under questions of different difficulty levels, and their corresponding actions $a_t^{(1)}$ and $a_t^{(2)}$ at the same position, even if they share identical reward contexts, their expected returns (baselines) should differ, i.e., the expected return is typically lower for more challenging problems.

We measure a problem's difficulty by estimating how often it is solved correctly under the current sampling policy. Concretely, we sample multiple trajectories in parallel for each problem. The fraction of these trajectories that yield a correct final answer serves as the problem's accuracy. We then discretize this accuracy into separate bins, effectively grouping the problems according to their estimated difficulty. All trajectories belonging to problems within the same accuracy bin form a common subset.

Compared to using direct reward contexts alone, this accuracy-based grouping offers a more robust estimate of expected returns, problems in the same bin share similar success rates. Moreover, unlike a pre-defined difficulty grouping, these

Accuracy Range	Retained Questions	MATH500	AIME2024	AMC2023	College Math	Olympiad Bench	GSM8K	GaokaoEn2023	Average
$[0.1-0.7]$	1805	83.4	23.3	62.5	50.0	46.7	92.9	72.2	61.6
$[0.2-0.8]$	2516	82.6	23.3	70.0	49.8	45.3	92.4	70.1	61.9
$[0.3-0.9]$	4448	81.6	23.3	70.0	49.4	44.7	92.0	68.1	61.3
$[0-1]$	Full	80.6	26.7	67.5	50.0	43.0	91.4	67.0	60.9

Table 10: Comparison of question filtering accuracy selection.

bins adjust dynamically as the model's capabilities evolve. Building on this approach, we propose two accuracy-based baseline estimation methods for offline RL as follows.

D.1.1 Accuracy-Grouped Baseline With Position Group

Within each accuracy bin, we further split actions based on their position in the trajectory. Concretely, we consider all actions occurring at the same step index across trajectories in the same bin to be comparable, and we compute their average return to serve as the baseline. Thus, when we look up the baseline for a particular action at a given step in a trajectory, we use the average return of all actions taken at that same step index in all trajectories belonging to the same accuracy bin.

D.1.2 Accuracy-Grouped Baseline With Reward Context

We also propose combining accuracy-based grouping with reward-context grouping. The underlying assumption is that even if two actions share the same immediate reward context, their expected returns can differ if they originate from different difficulty bins. Generally, problems that are harder to solve exhibit lower expected returns. Consequently, we first bin the trajectories by accuracy, then further group them by common reward context. Within each sub-group, we average the returns of all relevant actions to obtain the baseline.

D.2 Offline RL Implementation Details

In each iteration of offline RL training, we generate multiple trajectories (e.g., eight) per prompt in parallel. We then apply prompt filtering, rejection sampling, accuracy-based baseline estimation, advantage computation, and policy updates. Implementation details follow.

D.2.1 Prompt Filtering

As we sample multiple trajectories for each prompt, we compute the accuracy of each prompt. We retain prompts whose accuracy falls within a predefined range.

Our ablation study on Qwen2.5-Math-7B shown in Table 10 confirms that filtering improves performance. The most stable results are obtained with an accuracy range of $[0.1, 0.7]$, suggesting that including moderately difficult samples enhances the model's reasoning capabilities.

D.2.2 Rejection Sampling

We discard any trajectory that does not follow the alternation pattern of solution and verification: $y = (s_1, v_1, \ldots, s_k, v_k)$. Additionally, we remove malformed trajectories such as $y = (s_1, s_2, v_1)$. To mitigate reward hacking due to excessively long outputs, we eliminate trajectories where $R(s_t) = 1$ and $R(v_t) = 1$ at timestep t, but further actions are taken at $t + 1$. Moreover, we discard trajectories containing more than 20 actions, as excessive action sequences can introduce instability and deviate from expected solution structures.

D.2.3 Loss Function

To determine the best offline baseline method, we conducted ablation studies on Qwen2.5-Math-7B shown in Table 11. We found that using the accuracy-grouped baseline with an additional division by position provides the most stable results. When computing advantages, we subtract both the baseline and a scaled relative policy term like Equation 5. Notably, we fix π_{ref} as the reference policy instead of being updated at each iteration.

D.2.4 Training Hyperparameter Settings

We use a batch size of 64, a maximum learning rate of 5×10^{-7}, and a KL penalty coefficient of 0.1. The maximum training sequence length is set to 8192. We apply a warm-up phase of 5 steps and a clipping range parameter of 0.2. We use BF16 model precision in all experiments.

E Demo Cases

To intuitively demonstrate the effectiveness of our proposed method, we present the model's inference examples after RL on the MATH500 and StrategyQA datasets in the Figure 6 and Figure 7.

Baseline Method	Datasets							Average
	MATH500	AIME2024	AMC2023	College Math	Olympiad Bench	GSM8K	GaokaoEn2023	
Based on reward context	82.4	26.7	65.0	50.1	46.1	92.9	71.2	62.1
Based on accuracy group with position	83.4	23.3	62.5	50.0	46.7	92.9	72.2	61.6
Based on accuracy group with reward context	82.4	23.3	67.5	49.3	45.8	93.3	71.2	61.8

Table 11: The performance of different baselines

Problem: Lizzy, Megan, Oscar, and Patrick each have x pieces of candy, where x is a positive integer. Unfortunately, Patrick is the only one of the four who likes candy. So Lizzy gives all her candy to Megan. Then Megan gives all the candy she now has (which includes the candy Lizzy gave her) to Oscar. Then Oscar gives all the candy he now has to Patrick.
Let P be the number of pieces of candy Patrick has in the end. How many of the following statements are true? (Assume that we do not know exactly what x is.)

(a) 2 can be a divisor of P.

(b) 2 must be a divisor of P.

(c) 3 can be a divisor of P.

(d) 3 must be a divisor of P.

(e) 4 can be a divisor of P.

(f) 4 must be a divisor of P.

Response:

Let's analyze each statement step-by-step, considering the transformation of the pieces of candy from Lizzy to Megan, then to Oscar, and finally to Patrick... Wait, let me recheck my solution. To verify the correctness of the answer, we need to consider... Therefore, the answer is incorrect. Let me try again. Given that Lizzy, Megan, Oscar, and Patrick each start with x pieces of candy... Wait, let me recheck my solution. To verify the solution, we will analyze the number of pieces of candy, ... Therefore, the answer is incorrect. Let me try again. First, let's break down the candy distribution process... Wait, let me recheck my solution. To verify the answer, I will assess the properties of the... Let me try again.

To solve this problem, let's denote the initial number of pieces of candy each person has as x... Wait, let me recheck my solution. To verify the answer, we need to evaluate the given statements based on the problem conditions.

Lizzy, Megan, Oscar, and Patrick each start with x pieces of candy. The total amount of candy remains constant as it is transferred among them. Ultimately, Patrick ends up with all the candy, totaling $4x$ pieces.

Let's examine each statement:

(a) 2 can be a divisor of P: This is true because if x is even, then $4x$ is even and divisible by 2.

(b) 2 must be a divisor of P: This is true because $4x$ is always divisible by 2, regardless of x.

(c) 3 can be a divisor of P: This is true if x is a multiple of 3, then $4x$ will also be a multiple of 3.

(d) 3 must be a divisor of P: This is false because if x is not a multiple of 3, then $4x$ will not be a multiple of 3.

(e) 4 can be a divisor of P: This is true as $4x$ is divisible by 4.

(f) 4 must be a divisor of P: This is true because $4x$ is always divisible by 4, regardless of x.

Five statements are true: (a), (b), (c), (e), and (f). The only false statement is (d).
Therefore, the answer is correct.

Figure 5: SFT data example.

F Other Discussion

F.1 Discussion on Potential Risk

We have carefully considered potential risks associated with our work and found no significant concerns. Our approach, focused on enhancing LLM reasoning through self-verification and self-correction, does not introduce malicious or harmful effects, privacy issues, or security threats. Additionally, it does not contribute to biases, fairness concerns, or environmental impact. We believe our work is safe for responsible use in research.

F.2 Use of AI Assistant

In this work, we utilized an AI assistant solely for the purpose of refining and polishing the language of the manuscript. The AI assistant was employed to improve clarity, flow, and overall readability, ensuring the text adhered to academic writing standards. It was not involved in any data analysis, experimentation, or formulation of ideas. All research design, methodology, results, and conclusions were developed independently by the authors. The use of the AI assistant was limited to language enhancement and did not influence the content or scientific integrity of the work.

Problem: There are an infinite number of vectors $\mathbf{v}$ such that

$$\begin{pmatrix} 1 \\ 2 \\ -5 \end{pmatrix} \times \mathbf{v} = \begin{pmatrix} 90 \\ 30 \\ 30 \end{pmatrix}.$$

Find the vector $\mathbf{v}$ that has the smallest magnitude.

Reponse: To find the vector $\mathbf{v}$ that satisfies the equation $\begin{pmatrix} 1 \\ 2 \\ -5 \end{pmatrix} \times \mathbf{v} = \begin{pmatrix} 90 \\ 30 \\ 30 \end{pmatrix}$ and has the smallest magnitude, we start by letting $\mathbf{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$. The cross product $\begin{pmatrix} 1 \\ 2 \\ -5 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ is given by:

$\begin{pmatrix} 1 \\ 2 \\ -5 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2z + 5y \\ -5x + z \\ 2x - y \end{pmatrix}$

We set this equal to $\begin{pmatrix} 90 \\ 30 \\ 30 \end{pmatrix}$, giving us the system of equations:

$\begin{cases} 2z + 5y = 90 \\ -5x + z = 30 \\ 2x - y = 30 \end{cases}$

We solve the third equation for y:
$y = 2x - 30$
Substitute $y = 2x - 30$ into the first equation:
$2z + 5(2x - 30) = 90 \implies 2z + 10x - 150 = 90 \implies 2z + 10x = 240 \implies z + 5x = 120 \implies z = 120 - 5x$
Now substitute $z = 120 - 5x$ into the second equation:
$-5x + (120 - 5x) = 30 \implies -10x + 120 = 30 \implies -10x = -90 \implies x = 9$
Using $x = 9$, we find y and z:
$y = 2(9) - 30 = 18 - 30 = -12$
$z = 120 - 5(9) = 120 - 45 = 75$

Thus, the vector $\mathbf{v}$ is $\begin{pmatrix} 9 \\ -12 \\ 75 \end{pmatrix}$. To confirm that this is the vector with the smallest magnitude, we note that the general solution to the system of equations is of the form

$\mathbf{v} = \begin{pmatrix} 9 \\ -12 \\ 75 \end{pmatrix} + t \begin{pmatrix} 1 \\ 2 \\ -5 \end{pmatrix}$ for some scalar t.

The magnitude of $\mathbf{v}$ is minimized when $t = 0$, which corresponds to the vector $\begin{pmatrix} 9 \\ -12 \\ 75 \end{pmatrix}$.

The magnitude is $\boxed{\begin{pmatrix} 9 \\ -12 \\ 75 \end{pmatrix}}$.

Wait, let me recheck my solution.
To verify the correctness of the given answer, we can use the properties of the cross product and the given vectors.

The cross product of two vectors $\mathbf{a}$ and $\mathbf{b}$ is orthogonal to both $\mathbf{a}$ and $\mathbf{b}$. Given that $\mathbf{a} = \begin{pmatrix} 1 \\ 2 \\ -5 \end{pmatrix}$ and $\mathbf{v}$ are such that their cross product is $\mathbf{c} = \begin{pmatrix} 90 \\ 30 \\ 30 \end{pmatrix}$, we can check if the given answer $\mathbf{v} = \begin{pmatrix} 9 \\ -12 \\ 75 \end{pmatrix}$ satisfies this property.

First, we check if $\mathbf{a}$ and $\mathbf{v}$ are orthogonal to $\mathbf{c}$ by taking the dot product:
1. $\mathbf{a} \cdot \mathbf{c} = (1)(90) + (2)(30) + (-5)(30) = 90 + 60 - 150 = 0$ 2. $\mathbf{v} \cdot \mathbf{c} = (9)(90) + (-12)(30) + (75)(30) = 810 - 360 + 2250 = 2700$
Since $\mathbf{v} \cdot \mathbf{c} \neq 0$, the given answer $\mathbf{v}$ is not orthogonal to $\mathbf{c}$, which means it cannot be the correct vector $\mathbf{v}$ that satisfies the cross product equation.
Therefore, the answer is incorrect.

Let me try again.

To find the vector $\mathbf{v}$ such that $\begin{pmatrix} 1 \\ 2 \\ -5 \end{pmatrix} \times \mathbf{v} = \begin{pmatrix} 90 \\ 30 \\ 30 \end{pmatrix}$, we start by letting $\mathbf{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$. The cross product $\begin{pmatrix} 1 \\ 2 \\ -5 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ is given by the determinant of the following matrix:

$\begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 1 & 2 & -5 \\ x & y & z \end{vmatrix} = \mathbf{i}(2z + 5y) - \mathbf{j}(z + 5x) + \mathbf{k}(y - 2x).$

This gives us the vector:
$\begin{pmatrix} 2z + 5y \\ -z - 5x \\ y - 2x \end{pmatrix}.$

We set this equal to $\begin{pmatrix} 90 \\ 30 \\ 30 \end{pmatrix}$, so we have the system of equations:

$\begin{cases} 2z + 5y = 90, \\ -z - 5x = 30, \\ y - 2x = 30. \end{cases}$

From the second equation, we solve for z:
$-z - 5x = 30 \implies z = -5x - 30.$
From the third equation, we solve for y:
$y - 2x = 30 \implies y = 2x + 30.$
Substitute $y = 2x + 30$ and $z = -5x - 30$ into the first equation:
$2(-5x - 30) + 5(2x + 30) = 90.$
Simplify the left side:
$-10x - 60 + 10x + 150 = 90 \implies 90 = 90.$
This is an identity, so the system is consistent and has infinitely many solutions. The general solution is:

$\mathbf{v} = \begin{pmatrix} x \\ 2x + 30 \\ -5x - 30 \end{pmatrix}.$

To find the vector $\mathbf{v}$ with the smallest magnitude, we minimize the expression for the magnitude of $\mathbf{v}$:
$\|\mathbf{v}\| = \sqrt{x^2 + (2x + 30)^2 + (-5x - 30)^2}.$
Simplify the expression inside the square root:
$x^2 + (2x + 30)^2 + (-5x - 30)^2 = x^2 + (4x^2 + 120x + 900) + (25x^2 + 300x + 900) = 30x^2 + 420x + 1800.$
To minimize $30x^2 + 420x + 1800$, we complete the square:
$30x^2 + 420x + 1800 = 30(x^2 + 14x + 60) = 30((x + 7)^2 - 49 + 60) = 30((x + 7)^2 + 11) = 30(x + 7)^2 + 330.$
The minimum value occurs when $x = -7$, and the minimum value of the expression is 330. Substituting $x = -7$ into the expression for $\mathbf{v}$, we get:

$\mathbf{v} = \begin{pmatrix} -7 \\ 2(-7) + 30 \\ -5(-7) - 30 \end{pmatrix} = \begin{pmatrix} -7 \\ -14 + 30 \\ 35 - 30 \end{pmatrix} = \begin{pmatrix} -7 \\ 16 \\ 5 \end{pmatrix}.$

Thus, the vector $\mathbf{v}$ that has the smallest magnitude is $\boxed{\begin{pmatrix} -7 \\ 16 \\ 5 \end{pmatrix}}$.

Wait, let me recheck my solution.
To verify the correctness of the given answer, we can use the properties of the cross product and the given vectors.

The cross product of two vectors $\mathbf{a}$ and $\mathbf{b}$ is orthogonal to both $\mathbf{a}$ and $\mathbf{b}$. Given that $\mathbf{a} = \begin{pmatrix} 1 \\ 2 \\ -5 \end{pmatrix}$ and $\mathbf{v}$ are such that their cross product is $\mathbf{c} = \begin{pmatrix} 90 \\ 30 \\ 30 \end{pmatrix}$, we can check if the given answer $\mathbf{v} = \begin{pmatrix} -7 \\ 16 \\ 5 \end{pmatrix}$ satisfies this property.

First, we check if $\mathbf{a}$ and $\mathbf{v}$ are orthogonal to $\mathbf{c}$ by taking the dot product:
1. $\mathbf{a} \cdot \mathbf{c} = (1)(90) + (2)(30) + (-5)(30) = 90 + 60 - 150 = 0$ 2. $\mathbf{v} \cdot \mathbf{c} = (-7)(90) + (16)(30) + (5)(30) = -630 + 480 + 150 = 0$
Since both dot products are zero, the given answer $\mathbf{v}$ is orthogonal to $\mathbf{c}$, which means it could be a correct vector $\mathbf{v}$ that satisfies the cross product equation.
Therefore, the answer is correct.

Figure 6: MATH500 Case.

Problem You are given the following facts:

1. Chlorine, when added to water, creates household bleach.

2. Household bleach is available at most dollar stores.

Based on the facts above, answer the following question. Your final answer should be either "True" or "False".
Problem: Can you buy chlorine at a dollar store?
Response To determine if you can buy chlorine at a dollar store, let's analyze the given facts step by step:

1. Fact 1 states that when chlorine is added to water, it creates household bleach.

2. Fact 2 states that household bleach is available at most dollar stores.

From Fact 2, we know that household bleach is available at dollar stores. However, Fact 1 does not provide any direct information about the availability of chlorine itself at dollar stores. It only tells us that chlorine, when added to water, produces household bleach.
Since Fact 1 does not provide information about the direct sale of chlorine, we cannot conclude that chlorine is available at dollar stores based solely on the given facts.
Therefore, the answer is: $\boxed{\text{False}}$
Wait, let me recheck my solution.
The answer provided is "False." To verify this, let's examine the logical structure of the given statements and the inferred conclusion.
The first statement asserts that when chlorine is added to water, it creates household bleach. The second statement claims that household bleach is available at most dollar stores.
From these two statements, we can infer that if household bleach is available at dollar stores, it is possible that chlorine, when added to water, could be used to create household bleach that is sold at dollar stores. However, the statements do not explicitly confirm the availability of chlorine at dollar stores.
The answer "False" implies that it is not possible to buy chlorine at a dollar store. Given that the statements do not provide information about the direct sale of chlorine, but rather the creation of bleach from chlorine, the answer "False" is not necessarily incorrect. It is possible that chlorine is not sold at dollar stores, but it could be used to create bleach that is sold there.
Therefore, the answer is incorrect.
Let me try again.
To determine if you can buy chlorine at a dollar store, let's analyze the given facts step by step.

1. Fact 1: Chlorine, when added to water, creates household bleach.

2. Fact 2: Household bleach is available at most dollar stores.

From Fact 2, we know that household bleach is a product that can be found at dollar stores. However, Fact 1 only states that chlorine, when added to water, creates household bleach. It does not provide any information about whether chlorine itself is sold at dollar stores.
Since the availability of chlorine at dollar stores is not mentioned in the facts provided, we cannot definitively conclude that chlorine is available at dollar stores based solely on the given information.
Therefore, the answer is: $\boxed{\text{False}}$
Wait, let me recheck my solution.
The answer provided is "False." To verify this, let's examine the logical structure of the given statements and the inferred conclusion.
The first statement asserts that when chlorine is added to water, it creates household bleach. The second statement claims that household bleach is available at most dollar stores.
From these two statements, we can infer that if household bleach is available at dollar stores, it is possible that chlorine, when added to water, could be used to create household bleach that is sold at dollar stores. However, the statements do not explicitly confirm the availability of chlorine at dollar stores.
The answer "False" implies that it is not possible to buy chlorine at a dollar store. Given that the statements do not provide information about the direct sale of chlorine, but rather the creation of bleach from chlorine, the answer "False" is not necessarily incorrect. It is possible that chlorine is not sold at dollar stores, but it could be used to create bleach that is sold there.
Therefore, the answer is incorrect.
Let me try again.
To determine whether you can buy chlorine at a dollar store, let's analyze the given facts step by step.

1. Fact 1: Chlorine, when added to water, creates household bleach.

2. Fact 2: Household bleach is available at most dollar stores.

From Fact 2, we know that household bleach is available at most dollar stores. Since household bleach is made by adding chlorine to water, it implies that the process of making household bleach involves chlorine. However, Fact 1 does not provide any information about the direct sale of chlorine itself at dollar stores.
Given that household bleach is available at dollar stores and it is made from chlorine, it is reasonable to infer that chlorine is also available at dollar stores, as it is a necessary component for making household bleach.
Therefore, based on the given facts, it is reasonable to conclude that you can buy chlorine at a dollar store.
$\boxed{\text{True}}$
Wait, let me recheck my solution.
The answer provided is "True." To verify this, let's examine the logical structure of the given statements and the inferred conclusion.
The first statement asserts that when chlorine is added to water, it creates household bleach. The second statement claims that household bleach is available at most dollar stores.
From these two statements, we can infer that if household bleach is available at dollar stores, it is possible that chlorine, when added to water, could be used to create household bleach that is sold at dollar stores. This inference supports the conclusion that chlorine can be bought at a dollar store, as it is a necessary component for producing household bleach.
Therefore, the answer is correct.

Figure 7: StrategyQA Case.