Incentivizing LLMs to Self-Verify Their Answers

Fuxiang Zhang[1,2] **Jiacheng Xu**[1,2] **Chaojie Wang**[2] **Ce Cui**[2] **Yang Liu**[2] **Bo An**[1,2]*

[1] Nanyang Technological University, Singapore [2] Skywork AI

Abstract

Large Language Models (LLMs) have demonstrated remarkable progress in complex reasoning tasks through both post-training and test-time scaling laws. While prevalent test-time scaling approaches are often realized by using external reward models to guide the model generation process, we find that only marginal gains can be acquired when scaling a model post-trained on specific reasoning tasks. We identify that the limited improvement stems from distribution discrepancies between the specific post-trained generator and the general reward model. To address this, we propose a framework that incentivizes LLMs to self-verify their own answers. By unifying answer generation and verification within a single reinforcement learning (RL) process, we train models that can effectively assess the correctness of their own solutions. The trained model can further scale its performance at inference time by verifying its generations, without the need for external verifiers. We train our self-verification models based on Qwen2.5-Math-7B and DeepSeek-R1-Distill-Qwen-1.5B, demonstrating their capabilities across varying reasoning context lengths. Experiments on multiple mathematical reasoning benchmarks show that our models can not only improve post-training performance but also enable effective test-time scaling. Our code is available at `https://github.com/mansicer/self-verification`.

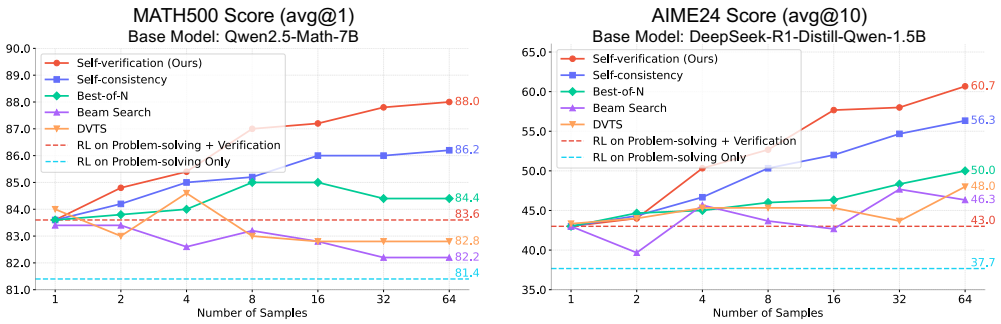


Figure 1: Average performance of post-trained models (dotted lines) and test-time scaling methods (solid lines) on the MATH500 and AIME24 benchmarks. Our self-verification framework not only enhances post-training performance with RL on both problem-solving and verification, but also enables effective test-time scaling with increased generation numbers by verifying its own solutions.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in a wide range of natural language tasks [1–3], particularly excelling in complex reasoning challenges such as mathematics

*Corresponding author. Email: `boan@ntu.edu.sg`

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

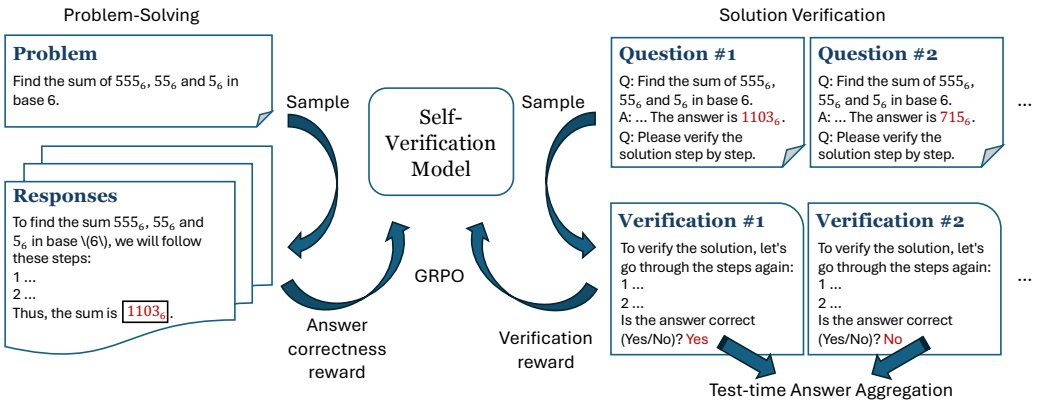


Figure 2: The framework of our self-verification framework. The model is trained to solve mathematical reasoning problems and verify generated solutions simultaneously.

and logic puzzles [4, 5]. For these verifiable tasks with ground-truth answers, researchers have proposed scaling laws in both post-training and test-time reasoning that substantially enhance LLM performance [6]. On the one hand, reinforcement learning (RL) has been widely adopted to align LLMs by rewarding predictions that match gold answers [7]. On the other hand, researchers leverage additional computational resources at inference time to improve accuracy through ensemble approaches [8, 9] and dedicated reward models (RMs) as verifiers [10–12].

Although both post-training and test-time scaling laws have significantly advanced the performance of LLMs, we observe that their combination yields limited synergistic benefits. Post-trained models are often specifically tuned on training data, resulting in a narrow generation distribution, which is often unrecognized by external verifiers trained on regular data. This distribution shift may lead to incorrect solution verifications and thus hinders test-time scaling performance [13]. As illustrated in Figure 1, we find that commonly used test-time scaling methods such as best-of-N and beam search with external RMs offer minimal improvements compared to the simple self-consistency method through majority voting. It remains challenging to synergize post-training and test-time scaling within an effective framework.

In this work, we propose to leverage *self-verification* to bridge the gap between post-training and test-time scaling. As illustrated in Figure 2, the core idea of self-verification is straightforward: we train the model not only to generate answers but also to verify the correctness of its solutions. Inspired by previous generative verifier works [14–16], we treat the pretrained LLM as a verifier, and then adopt RL algorithms to incentivize the LLM for self-verification, using reward signals derived from ground-truth answer correctness. We further design an online policy-aligned buffer and dynamic verification rewards to stabilize the joint training of answer generation and verification. The online buffer ensures that the input distribution of the verifier remains aligned with the latest model outputs, while the dynamic reward function utilizes multiple rollouts from the GRPO algorithm [17, 7] to automatically adjust reward signals, optimizing performance on challenging verification tasks. At inference time, we leverage the post-trained model for both answer generation and verification and adopt a weighted answer aggregation based on verification scores. Thanks to the unified model, our test-time scaling approach can be easily deployed within existing LLM inference engines, without relying on external RMs. Our experiments on mathematical reasoning benchmarks demonstrate that incorporating self-verification into post-training not only improves model performance but also enables effective test-time scaling with increasing generation budgets.

2 Preliminaries

2.1 GRPO for Math Reasoning Tasks

Reinforcement Learning (RL) has proven to be an effective approach for post-training a pretrained LLM with feedback from humans [1], AI proxies [18, 19], or ground-truth verification [20, 17]. To model the LLM generation process as a Markov Decision Process (MDP), we denote the language model as a policy $\pi_\theta(y \mid x)$, where θ is the model parameters, x is the input query and $y =$

$\{y_1, y_2, \ldots, y_T\}$ is the generated output sequence. The policy π_θ can be decomposed into a sequence of conditional probability distributions over tokens $\pi_\theta(y \mid x) = \prod_{t=1}^{T} P(y_t \mid y_{<t}, x; \theta)$, given the previous tokens.

To train policy π_θ with RL, we also need to define a reward function to provide training signals. For our math reasoning tasks, we follow the previous DeepSeek-R1 [7] work to adopt the correctness of the generated answer as the reward, evaluated by a rule-based verifier. Suppose the ground-truth answer to problem x is a^*. The correctness reward r_c can be defined as $r_c = \mathbb{I}[\mathcal{E}(y) = a^*]$, where $\mathcal{E}$ is a rule-based extractor that extracts the answer from a solution text and $\mathbb{I}$ is the indicator function with 0-1 binary output.

We use Group Relative Policy Optimization (GRPO) [17, 7] to train the policy π_θ. GRPO is a variant of Proximal Policy Optimization (PPO) [21, 22] that uses group-based policy sampling to compute the advantage as the training signal, providing higher efficiency and stability without introducing additional value functions. For a given input x, GRPO generates a group of answers $y_1, \ldots, y_G$ and optimizes the policy according to the estimated advantage derived from all responses in this group:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{x \in \mathcal{D}, \{y_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot|x)} \left[\frac{1}{G} \sum_{i=1}^{G} \left(\min \left(\frac{\pi_\theta(y_i|x)}{\pi_{\theta_{\text{old}}}(y_i|x)} A_i, \text{clip} \left(\frac{\pi_\theta(y_i|x)}{\pi_{\theta_{\text{old}}}(y_i|x)}, 1-\epsilon, 1+\epsilon \right) A_i \right) \right) - \beta \mathbb{D}_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right], \tag{1}$$

where $\mathcal{D}$ is the dataset of math questions, π_{old} is the old policy parameters for sampling, ϵ and β are two hyper-parameters, and $\mathbb{D}_{\text{KL}}(\cdot\|\cdot)$ is the KL-divergence. The advantage A_i of the generation y_i is defined as

$$A_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)}, \tag{2}$$

where r_i is the reward on response y_i, which can be the correctness reward r_c for standard GRPO.

2.2 Generative Verifiers

LLM-based verifiers are widely used for LLM post-training and scaling LLM performance at inference time. Traditional verifiers are typically trained as discriminative classifiers to score the generated solution with additional network heads. Recently, generative verifiers [14–16] propose to train the verifier through token prediction, in the same way as classical text generation. Unlike discriminative verifiers, generative verifiers can synthesize verification rationales for improved explainability.

A typical generative verifier will prompt the model to verify the solution y for a given problem x. The model is required to answer `Yes` or `No` after a template I like `Is the answer correct? (Yes/No)`. We also enable the model to generate intermediate rationales to justify its verification before outputting the final answer. This chain-of-thought (CoT) generation process has proven helpful for the final verification result.

At inference time, to better measure the quality of each solution, we use the likelihood of the `Yes` token as the solution score. Let y_v be the next token after the given template I, we have

$$s(x, y) = P(y_v = \texttt{Yes} \mid x, y, I). \tag{3}$$

In the original generative verifier work [14], the model is trained through supervised fine-tuning (SFT) with well-constructed data. Our work considers adopting RL to train such a verifier model within the training of math reasoning.

3 Self-Verification Framework

Since conventional test-time scaling methods often fail to generalize effectively for post-trained models, we introduce a *self-verification* framework that enables both efficient multi-task reinforcement learning (RL) and robust test-time scaling. In Section 3.1, we detail how answer generation and verification can be unified within a single RL process. Subsequently, in Section 3.2, we demonstrate how the trained model can leverage its verification capabilities to further enhance performance at inference time.

Algorithm 1 GRPO with Self-Verification

Input: Dataset $\mathcal{D}$, initial policy π_θ, online buffer size T_b, group size G, total training steps T
Initialize: Policy-aligned buffer $\mathcal{B} \leftarrow \emptyset$
for $t = 1, \ldots, T$ **do**
  Sample a data batch from the joint dataset $\mathcal{D} \cup \mathcal{B}$
  for input prompt x in the batch **do**
    Generate a group of responses $\{y_i\}_{i=1}^{G}$ from π_θ
    if the input x is from $\mathcal{D}$ **then**
      ▷ Math reasoning problem
      Compute correctness rewards r_c for each generation $y_1, \ldots, y_G$
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{(x, y_1), \ldots, (x, y_G)\}$
    else
      ▷ Verification problem
      Compute verification reward $\hat{r}_v$ for each generation $y_1, \ldots, y_G$ by Equation (5)
    end if
  end for
  Compute the total reward $r = r_c + \hat{r}_v$ for each sample
  Update the model π_θ according to Equation (1)
  Remove data samples from $\mathcal{B}$ that were collected before $t - T_b$ steps
end for

3.1 RL with Self-Verification

We adopt the GRPO algorithm [17], as described in Section 2.1, to train large language models (LLMs) on math reasoning tasks. This approach utilizes a correctness reward $r_c(x, y)$ provided by a rule-based verifier. To extend the model's capabilities of verification, we need to introduce an additional reward function. Since we have access to the ground-truth correctness for each generated solution y, we can directly compare the model's predicted verification outcome with the ground truth:

$$r_v(x, y) = \begin{cases} 1, & \text{if } \mathbb{I}(y_v = \texttt{Yes}) = r_c(x, y), \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Here, the token y_v is selected from `Yes` or `No`, representing the model's judgment on the verification task. Thus, r_v serves as a direct training signal for the verifier. Notably, the data for the verification task depends not only on the original math problem but also on the model's generated response, so we use an online data buffer to store real-time model responses during training.

Policy-Aligned Buffer. We leverage solutions generated during math reasoning training as data to train the verifier. However, selecting appropriate verification data poses a challenge. Retaining all historically generated solutions would force the verifier to learn from potentially meaningless or unrefined solutions produced during early training stages. To address this issue, we emphasize that our verifier should primarily focus on solutions representative of the current model's capabilities. Therefore, we implement a policy-aligned buffer $\mathcal{B}$ that stores only the most recent solutions, mitigating the risks associated with off-policy data. Specifically, we maintain only solutions generated within the last T_b training steps, ensuring the verification dataset is aligned with the evolving policy.

Dynamic Verification Reward. When training the verifier through RL, we find that a simple binary reward as defined in Equation (4) proves ineffective. As the model improves during training, we find that the correct solutions will soon dominate the verification dataset, creating an imbalanced data input. We want our verifier to focus on difficult verification problems during training. Hence, we introduce a dynamic verification reward that identifies challenging verification cases and dynamically adjusts rewards according to their difficulty. Leveraging the GRPO algorithm, we can obtain a group of generated responses $y_1, \ldots, y_G$ for each problem x. Denoting G_c and G_i as the number of correct and incorrect generations in the group, we define the dynamic verification reward $\hat{r}_v$ as:

$$\hat{r}_v(x, y) = \begin{cases} \frac{2G_i}{G}, & \text{if } y_v = \texttt{Yes} \text{ and } r_c(x, y) = 1, \\ \frac{2G_c}{G}, & \text{if } y_v = \texttt{No} \text{ and } r_c(x, y) = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

This reward design provides higher incentives when the verifier successfully identifies a correct solution to a difficult problem or an incorrect solution for an easy problem. We maintain the expected reward for correct verification at 1, ensuring it remains comparable in scale to the problem-solving reward r_c.

As detailed in Algorithm 1, our algorithm integrates self-verification into GRPO training by maintaining an online buffer of recent model generations. For each training step, we combine the original dataset and the buffer to sample problems. When processing dataset problems, we generate multiple solutions, compute their correctness rewards, and add them to the buffer. For problems from the buffer, we compute dynamic verification rewards based on the difficulty of verification. The final reward combines both correctness and verification components. This approach ensures the model learns to generate correct solutions while simultaneously developing verification capabilities.

3.2 Self-Verification for Test-time Scaling

At inference time, as our post-trained model possesses a strong built-in verifier, we can scale the model performance by allowing the model to verify its own solutions. Given a generation budget of N, we first sample N candidate responses $y_1, \ldots, y_N$ from the model. For each response, we compute a verification score $s(x, y_i)$ as defined in Equation (3). We then extract the set of unique answers $\mathcal{A} = \{\mathcal{E}(y_1), \ldots, \mathcal{E}(y_N)\}$. We further aggregate the verification scores for each candidate answer to determine the final prediction $\hat{a}$:

$$\hat{a} = \text{argmax}_{a \in \mathcal{A}} \sum_{i=1}^{N} \mathbb{I}[\mathcal{E}(y_i) = a] \left(1 + \alpha s(x, y_i)\right), \tag{6}$$

where α is a hyperparameter that controls the scale of verification scores. When $\alpha = 0$, this reduces to standard majority voting; as $\alpha \to \infty$, the answer with the highest verification score dominates. By choosing a moderate value for α, we can achieve a balance between consensus and confidence, leading to more robust and reliable predictions. A key advantage of this self-verification approach is its efficiency: it requires only a single LLM deployment at inference time, eliminating the need for external verifier models. This makes our framework highly practical and easily compatible with modern LLM inference engines such as vLLM [23] and SGLang [24].

4 Related Work

RL for Reasoning. The OpenAI o1 model series [6] has spurred a surge of research into the reasoning capabilities of large language models (LLMs). Early self-improvement methods [25–27] enable models to iteratively generate and refine their own reasoning paths by selecting high-quality solutions. With the emergence of DeepSeek-R1 [7], RL has become increasingly prominent as a means to directly incentivize solution correctness. Recent works including SimpleRL-Zoo [28], DeepScaleR [29], Open R1 [30], and Light-R1 [31] demonstrate substantial improvements on reasoning tasks by applying RL to post-train pretrained models. Building on these advances, our work extends RL-based approaches to jointly train the model both for problem-solving and verification, enabling simultaneous improvement in these two tasks.

Test-time Scaling. Numerous techniques have been developed to enhance LLM performance at inference time by increasing computational resources and promoting output diversity. Self-consistency decoding, for example, samples multiple solutions and aggregates them via majority voting [8], while some works also consider scaling the response length for more exhaustive reasoning paths [32]. Other approaches often leverage external reward models (RMs) to guide generation, including beam search [33, 34], Monte-Carlo tree search [35–37], multi-turn correction [38–42]. For general pretrained models, process-based supervision—which rewards each correct reasoning step—has been shown to further improve performance [11, 12, 43, 33]. However, our findings indicate that these process-based methods may be less effective for specifically post-trained models due to severe distributional shifts.

Verifier Training. Robust verification mechanisms are crucial for reliable reasoning in LLMs. Early approaches typically train discriminative verifiers [44] by adding an auxiliary classification head to the model, with parameters learned from pairwise feedback [1, 18] or ground-truth labels [45, 46]. Inspired by the LLM-as-a-judge paradigm [47] used in the evaluation, recent work has explored generative verification, where the model produces verification results through text generation

Table 1: Average **greedy-decoding scores** of different models on math reasoning benchmarks after post-training. The best scores from each model series are highlighted in bold. For AIME24, AIME25, and AMC23, we report the average scores over 10 samples for each problem.

Model	MATH500	AIME24 (avg@10)	AIME25 (avg@10)	AMC23 (avg@10)	Olympiad Bench
Model Series: Qwen2.5-Math-7B					
Self-Verification-Qwen-7B (Ours) (Problem-solving + verification)	**83.60**	20.00	**16.67**	63.75	**34.81**
Qwen2.5-Math-7B (Base model)	62.00	14.67	5.00	45.25	17.63
GRPO-Qwen-7B (Problem-solving Only)	81.40	19.67	15.67	**65.50**	32.89
SimpleRL-Qwen-Math-7B ([28])	80.80	**23.33**	10.00	63.75	32.15
Model Series: DeepSeek-R1-Distill-Qwen-1.5B					
Self-Verification-R1-1.5B (Ours) (Problem-solving + verification)	**87.00**	**43.00**	**31.33**	**77.50**	**44.30**
R1-Distill-Qwen-1.5B (Base model)	80.00	24.33	25.00	64.25	32.89
GRPO-R1-1.5B (Problem-solving only)	**87.00**	37.67	26.67	72.50	40.74
DeepScaleR-1.5B-Preview ([29])	83.00	37.00	31.00	77.25	43.56

[14–16]. These generative verifiers can articulate chain-of-thought rationales [48] when assessing solutions, thereby making the verification process more transparent and interpretable. Previous works usually train generative verifiers via supervised fine-tuning (SFT) on datasets containing high-quality verification responses, whereas a concurrent work [49] also considers using SFT to tune a verifier model within varying RL training algorithms. To our knowledge, our work is the first to propose a unified RL framework that unifies problem-solving and verification.

5 Experiments

In this section, we aim to examine the effectiveness and characteristics of our self-verification framework. We compare our methods with standard RL and test-time baselines to answer the following questions: (1) How does imposing self-verification on the RL process compare to standard RL for problem-solving? (2) Can the learned model verify its own solutions better than external verifiers? (3) How does self-verification help with test-time scaling? (4) Is the self-verification process efficient enough for post-training and test-time scaling?

Models and Training. In our experiments, we primarily use two pretrained models for RL training with self-verification, which are Qwen2.5-Math-7B [50] and DeepSeek-R1-Distill-Qwen-1.5B [7]. The Qwen2.5-Math-7B model is a 7B-parameter model pretrained in massive math data with a relatively short generation length, while DeepSeek-R1-Distill-Qwen-1.5B is a distilled version of the powerful DeepSeek-R1 model with the ability of generating long CoT responses. We use the popular `verl` framework [51] as the code base for RL training, where the maximal context length for Qwen2.5-Math-7B is 4k and for DeepSeek-R1-Distill-Qwen-1.5B is 16k. We name the trained models as Self-Verification-Qwen-7B and Self-Verification-R1-1.5B, respectively. We provide our training details in Appendix B.

Data and Benchmarks. We target our self-verification framework on mathematical reasoning tasks. Considering previous popular implementations of GRPO on math reasoning [28, 29], we use the level 3-5 data from the math training dataset [4] to train the Qwen2.5-Math-7B model and a combined math reasoning dataset sorted by DeepScaleR [29] to train the DeepSeek-R1-Distill-Qwen-1.5B model. For the evaluation benchmarks, we adopt the MATH500 dataset [10], AIME 2024 and 2025 problems, AMC 2023 problems, and OlympiadBench [52], which are commonly used benchmarks

Table 2: The performance of different models on verifying MATH500 solutions generated by the Self-Verification-Qwen-7B model. We highlight the best scores from the open-source models in bold.

Category	Method	Accuracy	F1 Score
Open-source Models (~7B)	Self-Verification-Qwen-7B (Ours)	**87.20**	**92.83**
	Qwen2.5-Math-7B (Base model)	73.20	84.93
	Llama-3.1-8B-Instruct	67.00	78.20
Proprietary Models	GPT-4o	85.20	91.57
	Claude-3.7-Sonnet	90.20	94.46
	DeepSeek-v3	89.00	93.73

Table 3: The performance of different models on verifying AIME24 solutions generated by the Self-Verification-R1-1.5B model. We highlight the best scores from the open-source models in bold.

Category	Method	Accuracy	F1 Score
Open-source Models (1.5B or ~7B)	Self-Verification-R1-1.5B (Ours)	**56.67**	**67.72**
	R1-Distill-Qwen-1.5B (Base model)	38.00	49.46
	R1-Distill-Qwen-7B	46.00	59.50
	Llama-3.1-8B-Instruct	55.67	45.71
Proprietary Models	GPT-4o	59.33	65.54
	Claude-3.7-Sonnet	64.33	71.16
	DeepSeek-v3	57.67	66.67

for evaluating math reasoning models [50]. We enable a context length of 4k for methods based on Qwen-2.5-Math-7B and a context length of 16k for methods based on DeepSeek-R1-Distill-Qwen-1.5B. Unless stated otherwise, we use the accuracy of the generated answers as the score shown in the experimental results, which is automatically verified by the `math-verify` library from HuggingFace[2]. As the evaluation data is relatively scarce in AIME24, AIME25, and AMC23, we repeat each problem 10 times and use the average accuracy as the result.

Baselines. Our baselines include standard RL methods and test-time scaling baselines. For the RL algorithms, we compare the standard GRPO version without self-verification training and name them GRPO-Qwen-7B and GRPO-R1-1.5B, respectively. To compare the validity of our RL process with prior works, we also include SimpleRL-Qwen-Math-7B model [28] and DeepScaleR-1.5B-Preview model [29] as our baselines, which are also tuned based on Qwen2.5-Math-7B and DeepSeek-R1-Distill-Qwen-1.5B. For the test-time scaling baselines, we include the following methods:

- **Self-consistency** [8] uses multiple CoT solutions to improve the accuracy of the final answer through majority voting.
- **Best-of-N** [33] adopts the sample with the highest score from N generated responses.
- **Process-based test-time scaling** methods adopt a process-based reward model (PRM) [10, 45] to further supervise the generation in each step with various strategies like **beam search** [43, 33] and **Diverse Verifier Tree Search (DVTS)** [53].

We can split the baselines into two categories. The self-consistency method is a simple ensemble method without using other models, while other baselines including best-of-N, beam search, and DVTS require external reward models to guide generations. For the choice of models, we follow Beeching et al. [53] to use the RLHFlow `Llama3.1-8B-PRM-Deepseek-Data` PRM with 8B parameters [45, 46], which is the best PRM revealed in their work with similar size to our Qwen-based 7B verifier model.

[2] `https://github.com/huggingface/Math-Verify`

Table 4: Average **test-time scaling** scores of different methods on various math reasoning benchmarks. All the test-time scaling methods have a budget of 16 samples for each problem. The best scores from each model series are highlighted in bold. For AIME24, AIME25, and AMC23, we report the average scores over 10 samples for each problem.

Method@16	MATH500	AIME24 (avg@10)	AIME25 (avg@10)	AMC23 (avg@10)	Olympiad Bench
Model Series: Qwen2.5-Math-7B					
Self-Verification (Ours)	**87.20**	**26.67**	19.00	**73.25**	**39.70**
Self-Consistency	86.00	23.67	**21.67**	71.25	39.11
Best-of-N	85.00	23.33	16.67	64.25	37.92
Beam Search	82.80	21.00	15.00	67.25	35.71
DVTS	82.80	21.67	20.33	67.50	35.85
Model Series: DeepSeek-R1-Distill-Qwen-1.5B					
Self-Verification (Ours)	**93.60**	**57.67**	**37.67**	**92.00**	**50.96**
Self-Consistency	91.00	52.00	36.67	87.50	47.56
Best-of-N	86.00	46.33	33.67	83.75	43.41
Beam Search	88.40	42.67	33.00	85.25	44.59
DVTS	90.80	45.33	32.33	82.50	45.18

5.1 Post-training Model Performance

Performance on Problem-Solving We begin by examining the impact of incorporating self-verification during post-training on the greedy-decoding performance of the model, without employing any additional test-time techniques. To this end, we evaluate our self-verification models across several math reasoning benchmarks, as summarized in Table 1. Both Self-Verification-Qwen-7B and Self-Verification-R1-1.5B consistently achieve strong results, outperforming their respective base models by a significant margin due to the reinforcement learning process. Notably, integrating self-verification into RL yields even higher scores than the standard GRPO models, suggesting that self-verification intrinsically enhances the model's problem-solving abilities. We speculate that this synergy arises because the verification task encourages the model to develop a deeper understanding of the problem's logical structure, which in turn enhances its problem-solving capabilities. Although no new input is introduced, verification training elicits knowledge and reasoning patterns that generalize back to the generation task.

Performance on Verifying Solutions We further evaluate the effectiveness of our self-verification models in verifying their own generated solutions. Specifically, we prompt various LLMs to verify the solution for a given problem, extract their binary verification answer (`Yes` or `No`), and compare it against the ground-truth correctness. As presented in Table 2 and Table 3, we conduct this generative verification procedure with multiple LLMs, including open-source models of comparable size to ours as well as proprietary models accessed via APIs. On the MATH500 and AIME24 benchmarks, our self-verification models demonstrate substantial improvements in both accuracy and F1 score over similarly sized open-source baselines. Notably, our models also achieve performance on par with leading commercial systems such as GPT-4o [2], Claude-3.7-Sonnet [3], and DeepSeek-v3 [54]. Our model can even surpass GPT-4o in verifying their own responses despite having significantly fewer parameters. We also observe that verifying AIME24 solutions poses a greater challenge for all models, likely due to the increased complexity of the problems.

5.2 Self-verification on Test-time Scaling

As stated in Section 3.2, our trained self-verification model can scale its performance by verifying its generated answers. We use Self-Verification-Qwen-7B and Self-Verification-R1-1.5B to generate solutions with a given sample budget and further adopt different test-time scaling methods to evaluate the performance of the final answer. In Figure 1, we show the scaling performance of different methods when sampling from Self-Verification-Qwen-7B on MATH500 and Self-Verification-R1-1.5B on AIME24. The results show that test-time scaling with self-verification yields the best performance among all methods. Maybe surprisingly, we find that the simple self-consistency

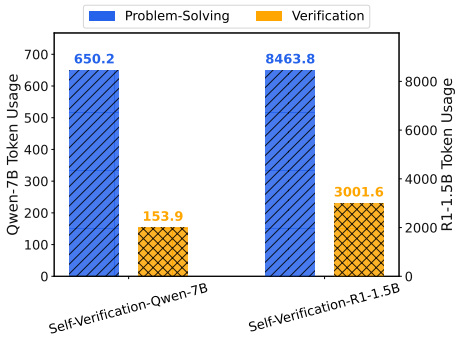


Figure 3: Token usage comparison between problem-solving and verification tasks.

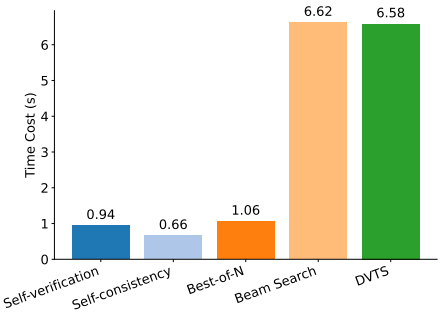


Figure 4: Average time cost of different test-time scaling methods per problem from MATH500.

Table 5: Ablation of the aggregation coefficient α on MATH500 with a 16-sample budget. Accuracy is reported for different α settings when selecting answers using verification probabilities.

Self-Verification-Qwen-7B@16	
α	**Accuracy**
0.3	86.60
1.0	**87.20**
3.0	87.10
5.0	87.10

Self-Verification-R1-1.5B@16	
α	**Accuracy**
0.03	92.60
0.10	**93.60**
0.30	92.40
1.00	91.00

method also presents competitive performance by majority voting. In contrast, methods based on external verifiers like best-of-N, beam search, and DVTS perform worse than self-consistency. This observation proves our hypothesis that external verifiers trained on specific data do not generalize to verifying our post-trained models. Process-based methods like beam search and DVTS adopt a step-wise generation, which may further amplify the distribution shift during the generation process. In Table 4, we show the detailed results on more math reasoning benchmarks with a generation budget of 16, which is a common number of generations balancing between the quality and the efficiency. We see that our self-verification models can consistently outperform all other methods on most tasks.

5.3 Analysis on the Efficiency of Self-Verification

In the post-training stage, our framework is comparable in training cost to other RL methods. As the verification process is expected to be less difficult than problem-solving, we find that the average tokens the model spends on verification is actually lower for both short-context Qwen 7B model and long-context R1 distilled model. As shown in Figure 3, the average token usage in the verification task is 24% of the problem-solving task for Self-Verification-Qwen-7B and 35% for Self-Verification-R1-1.5B. The lower token usage makes our test-time scaling method more efficient since the verification process takes much less time than the problem-solving process. Figure 4 illustrates the average time cost per problem on the MATH500 benchmark for various test-time scaling methods, evaluated on a fixed hardware platform. Unlike approaches that rely on external Reward Models (RMs), our unified model architecture requires serving only a single LLM, which reduces overhead and improves efficiency. Our self-verification method is faster than best-of-N sampling and demonstrates significant efficiency gains over process-based methods like beam search and DVTS. Although its time cost is slightly higher than simple majority voting, this represents a favorable trade-off for improved performance. We report the implementation details of our inference-time scaling in Appendix C.

Ablation Studies We investigate how the mechanisms and hyperparameters proposed in our methodology affect the post-training and test-time performance. First, we examine the test-time aggregation coefficient α, which balances majority voting with verifier confidence. We show the MATH500 performance of our two self-verification models in Table 5. The Self-Verification-Qwen-7B model peaks at $\alpha = 1.0$ while not showing significant performance degradation across different α

settings. In contrast, Self-Verification-R1-1.5B is more sensitive to this hyperparameter, preferring a smaller $\alpha = 0.1$, likely due to its small model size that limits the verification capability. We also report the ablation results on our two post-training mechanisms, including the policy-aligned buffer and the dynamic verification reward, in Appendix F.

6 Conclusion and Limitations

In this paper, we introduced a novel self-verification framework for enhancing the mathematical reasoning of LLMs. Our approach integrates verification capabilities directly into the problem-solving model through a specialized RL process, enabling models to evaluate their own solutions in a generative verification process. Through extensive experiments across multiple mathematical benchmarks, we demonstrated that our self-verification models not only achieve superior performance in problem-solving tasks but also excel at verifying solution correctness. Our self-verification framework addresses the distribution shift issue that arises during test-time scaling with external verifiers, achieving better test-time scaling performance compared to other baselines.

Our work has several limitations. First, the self-verification framework is primarily tailored for mathematical reasoning tasks, and its direct applicability to other domains such as code generation and agentic tasks, remains to be tested. Adapting self-verification to these areas may require additional task-specific designs, integration with external tools, or new verification strategies. Second, our test-time scaling by self-verification focuses on aggregating answers from multiple generations but does not consider the scale of response length. A multi-turn response generation between problem-solving and verification may further improve inference-time performance, which is a promising direction to extend our framework. We discuss the broader impact of our work in Appendix A.

Acknowledgments and Disclosure of Funding

Bo An is supported by the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Programme (AISGAward No: AISG4-GC-2023-009-1B).

References

[1] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022.

[2] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *CoRR*, abs/2410.21276, 2024.

[3] Anthropic. Claude 3.7 Sonnet and Claude Code. `https://www.anthropic.com/news/claude-3-7-sonnet`, 2025. Accessed: 2025-05-14.

[4] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[5] Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. Logic-RL: Unleashing LLM reasoning with rule-based reinforcement learning. *CoRR*, abs/2502.14768, 2025.

[6] OpenAI. Learning to reason with LLMs, 2024. URL `https://openai.com/index/learning-to-reason-with-llms`. Accessed: 2025-05-05.

[7] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan

Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, and S. S. Li. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *CoRR*, abs/2501.12948, 2025.

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

[9] Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need. *CoRR*, abs/2402.05120, 2024.

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

[11] Jonathan Uesato, Nate Kushman, Ramana Kumar, H. Francis Song, Noah Y. Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *CoRR*, abs/2211.14275, 2022.

[12] Seungone Kim, Ian Wu, Jinu Lee, Xiang Yue, Seongyun Lee, Mingyeong Moon, Kiril Gashteovski, Carolin Lawrence, Julia Hockenmaier, Graham Neubig, and Sean Welleck. Scaling evaluation-time compute with reasoning models as process evaluators. *CoRR*, abs/2503.19877, 2025.

[13] Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. Inference scaling fLaws: The limits of LLM resampling with imperfect verifiers. *CoRR*, abs/2411.17501, 2024.

[14] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative Verifiers: Reward modeling as next-token prediction. *CoRR*, abs/2408.15240, 2024.

[15] Dakota Mahan, Duy Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato, Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. Generative Reward Models. *CoRR*, abs/2410.12832, 2024.

[16] Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. Inference-time scaling for generalist reward modeling. *CoRR*, abs/2504.02495, 2025.

[17] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024.

[18] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosiute, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemí Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional AI: Harmlessness from AI feedback. *CoRR*, abs/2212.08073, 2022.

[19] Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. RLAIF vs. RLHF: Scaling reinforcement learning from human feedback with AI feedback. In *International Conference on Machine Learning*, 2024.

[20] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. WizardMath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *CoRR*, abs/2308.09583, 2023.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[22] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations*, 2016.

[23] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Symposium on Operating Systems Principles*, pages 611–626, 2023.

[24] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

[25] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, 2022.

[26] Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron C. Courville, Alessandro Sordoni, and Rishabh Agarwal. V-STaR: Training verifiers for self-taught reasoners. *CoRR*, abs/2402.06457, 2024.

[27] Jing-Cheng Pang, Pengyuan Wang, Kaiyuan Li, Xiong-Hui Chen, Jiacheng Xu, Zongzhang Zhang, and Yang Yu. Language model self-improvement by reinforcement learning contemplation. In *International Conference on Learning Representations*, 2024.

[28] Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. SimpleRL-Zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *CoRR*, abs/2503.18892, 2025.

[29] Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. DeepScaleR: Surpassing o1-Preview with a 1.5b model by scaling RL. `https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-\19681902c1468005bed8ca303013a4e2`, 2025. Notion Blog.

[30] Hugging Face. Open R1: A fully open reproduction of DeepSeek-R1, January 2025. URL `https://github.com/huggingface/open-r1`.

[31] Liang Wen, Yunke Cai, Fenrui Xiao, Xin He, Qi An, Zhenyu Duan, Yimin Du, Junchen Liu, Lifu Tang, Xiaowei Lv, Haosheng Zou, Yongchao Deng, Shousheng Jia, and Xiangzheng Zhang. Light-R1: Curriculum SFT, DPO and RL for long COT from scratch and beyond. *CoRR*, abs/2503.10460, 2025.

[32] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel J. Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *CoRR*, abs/2501.19393, 2025.

[33] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *CoRR*, abs/2408.03314, 2024.

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

[35] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023.

[36] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. *CoRR*, abs/2305.14992, 2023.

[37] Xidong Feng, Ziyu Wan, Muning Wen, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

[38] Xiaoyu Tian, Sitong Zhao, Haotian Wang, Shuaiting Chen, Yunjie Ji, Yiping Peng, Han Zhao, and Xiangang Li. Think twice: Enhancing LLM reasoning by scaling multi-round test-time thinking. *CoRR*, abs/2503.19855, 2025.

[39] Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. Small language models need strong verifiers to self-correct reasoning. In *Findings of the Association for Computational Linguistics*, pages 15637–15653, 2024.

[40] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal M. P. Behbahani, and Aleksandra Faust. Training language models to Self-Correct via reinforcement learning. In *International Conference on Learning Representations*, 2025.

[41] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

[42] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

[43] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *CoRR*, abs/2408.03314, 2024.

[44] 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. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021.

[45] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Annual Meeting of the Association for Computational Linguistics*, pages 9426–9439, 2024.

[46] Wei Xiong, Hanning Zhang, Nan Jiang, and Tong Zhang. An implementation of generative PRM. `https://github.com/RLHFlow/RLHF-Reward-Modeling`, 2024.

[47] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, 2023.

[48] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

[49] Kusha Sareen, Morgane M Moss, Alessandro Sordoni, Rishabh Agarwal, and Arian Hosseini. Putting the value back in RL: Better test-time scaling by unifying LLM reasoners with verifiers. *CoRR*, abs/2505.04842, 2025.

[50] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *CoRR*, abs/2409.12122, 2024.

[51] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *European Conference on Computer Systems*, pages 1279–1297, 2025.

[52] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Annual Meeting of the Association for Computational Linguistics*, pages 3828–3850, 2024.

[53] Edward Beeching, Lewis Tunstall, and Sasha Rush. Scaling test-time compute with open models, 2025. URL `https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute`.

[54] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. DeepSeek-V3 technical report. *CoRR*, abs/2412.19437, 2025.

[55] Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. Skywork open reasoner series. `https://capricious-hydrogen-41c.notion.site/Skywork-Open-Reaonser-Series-1d0bc9ae823a80459b46c149e4f51680`, 2025. Notion Blog.

[56] Zitian Gao, Yitong Shu, Jiacheng Li, Shizhe Liu, Zixuan Han, Wang Zhou, Lingpeng Zhang, and Yang Liu. One-shot entropy minimization. *CoRR*, abs/2505.20282, 2025.

A Broader Impact

Our self-verification framework for LLMs has several potential societal impacts, both positive and negative, that warrant discussion.

Positive Impacts The primary beneficial impact of our work is the improvement of LLM reliability in reasoning tasks. By enabling models to verify their own outputs, we can contribute to reducing hallucinations and factual errors in AI systems, which is crucial for applications in education, scientific research, and decision support. The self-verification approach also improves computational efficiency compared to methods requiring external verifiers, potentially reducing energy consumption and computational costs associated with deploying reasoning systems at scale. Additionally, our unified model approach demonstrates that complementary capabilities (reasoning and verification) can be trained together, which may inspire similar approaches in other domains requiring accuracy and trustworthiness.

Risks and Limitations Despite the improvements in verification capabilities, our approach does not eliminate the risk of incorrect outputs or the model confidently asserting wrong answers. There is a potential concern that users might place excessive trust in self-verified outputs, believing them to be more reliable than they actually are, particularly in high-stakes domains like healthcare or financial analysis. The verification capability is also currently limited to mathematical reasoning tasks, and the generalizability to other domains remains uncertain. Furthermore, the increased capabilities could potentially be misused in generating deceptive content that appears more legitimate due to self-verification signals.

Safeguards and Mitigations To address these concerns, we recommend several safeguards: (1) Clear communication to users about the limitations of self-verification, including transparency about error rates; (2) Complementary use of human verification for critical applications; (3) Continued research into detecting when models are operating outside their reliable domains; and (4) Development of benchmark tests to evaluate verification capabilities across diverse problem types and difficulty levels. We have also published our methodology transparently to enable further research into both the capabilities and limitations of self-verification approaches.

The overall aim of our work is to contribute to the development of more reliable and trustworthy AI systems, with self-verification representing one component in the broader ecosystem of techniques needed for responsible AI deployment.

B Technical Details on Post-training

In this section, we provide comprehensive details about our post-training process with self-verification. We use the popular verl framework[3], which is the open-source version of Sheng et al. [51], as our RL training framework. For the training resources of our models, we use two nodes of 8 NVIDIA GPUs with 80GB memory each. The detailed training configurations and hyperparameters are listed in Table 6.

Model and Data Configuration For Self-Verification-Qwen-7B, we use Qwen2.5-Math-7B as our base model and train on the MATH training data [4] with level 3 to 5 curated by SimpleRL-Zoo [28]. The model operates with a maximum prompt length of 2048 tokens and a maximum response length of 2048 tokens. For Self-Verification-R1-1.5B, we use DeepSeek-R1-Distill-1.5B as the base model and train on the comprehensive dataset from DeepScaleR [29], which is a collection from multiple math reasoning data sources. While keeping the same prompt length limit, we extend the maximum response length to 14336 tokens to accommodate the model's capability for generating longer chain-of-thought reasoning. Different from most math RL implementations, we use a longer prompt length of 2048 as we expect more input tokens for the verification task. For the Self-Verification-R1-1.5B model, we remove the thinking content from the original response, wrapped by the `<think>...</think>` tags, to reduce the input prompt length, which also follows the implementation of DeepSeek-R1 [7] when building multi-turn chat conversation.

[3] `https://github.com/volcengine/verl`

Table 6: GRPO hyperparameters on the post-training stage with self-verification

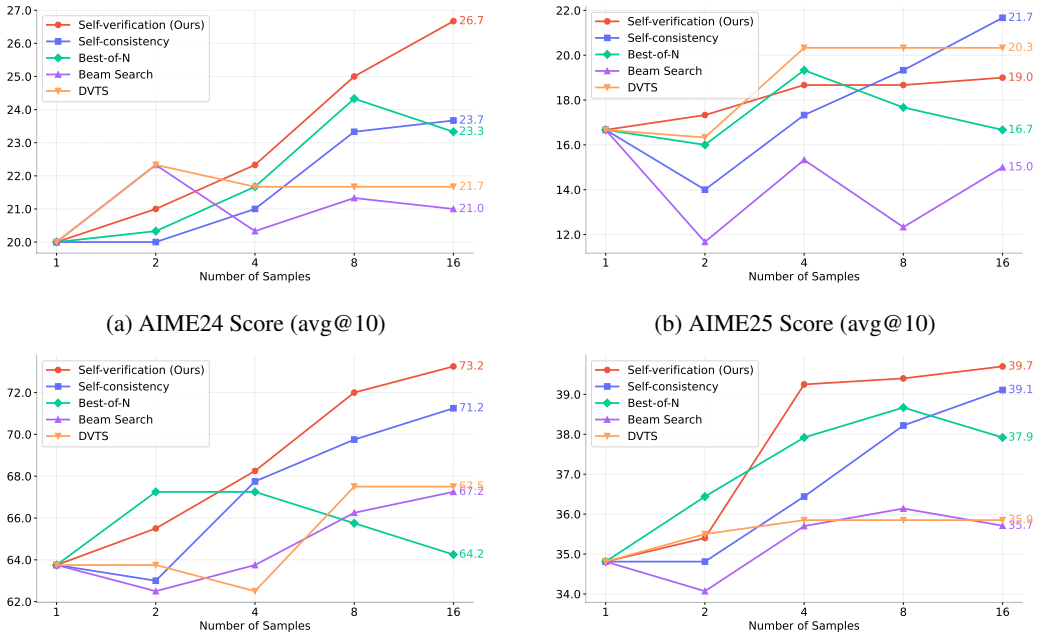
Hyperparameter	**Self-Verification-Qwen-7B**	**Self-Verification-R1-1.5B**
Model and Data		
Base Model	Qwen2.5-Math-7B	DeepSeek-R1-Distill-1.5B
Training Data	MATH-level3to5 [28]	DeepScaleR [29]
Max Prompt Length	2048	2048
Max Response Length	2048	14336
RL Hyperparameters		
Train Batch Size	128	128
PPO Mini-batch Size	64	64
Learning Rate	1e-6	1e-6
KL Loss Coefficient	0.001	0.001
Entropy Coefficient	0.001	0.005
Adaptive Entropy	False	True
Target Entropy	N/A	0.2
Entropy Coeff Delta	N/A	0.0001
Max/Min Entropy Coeff	N/A	0.005/0
Rollout Settings		
Engine	vLLM [23]	vLLM [23]
GPU Memory Utilization	0.8	0.8
GRPO Group Size	8	8
Temperature	0.6	0.6
Training Schedule		
Total Training Steps	1000	2000
Rejection Sampling	✓	✓
Online Data Buffer		
Data Buffer Size	5000	40000
Save Frequency T_b	60	60

RL Settings We adopt the GRPO algorithm for both models with the same batch size: a training batch size of 128 and a PPO mini-batch size of 64. Other training-related configurations are similar to the original implementation in verl. The learning rate is set to 1e-6 for both models. For the KL loss coefficient, we use 0.001 to maintain a balance between exploration and policy improvement. The entropy coefficient is set differently: 0.001 for Qwen-7B and 0.005 for R1-1.5B. Notably, we follow He et al. [55] to enable adaptive entropy for R1-1.5B with a target entropy of 0.2, allowing the entropy coefficient to adjust between 0 and 0.005 per step, as we find that a fixed entropy coefficient for the DeepSeek-R1-Distill-1.5B model results in an extremely unstable training process. We leverage the vLLM engine [23] for efficient inference during training, setting the GPU memory utilization to 0.8. For both models, we use a group size of 8 for GRPO and a temperature of 0.6 for sampling. The training runs for 1500 steps for Qwen-7B and 2000 steps for R1-1.5B, with rejection sampling enabled to filter out invalid generations.

Additional Rewards for Qwen-7B Model When using the Qwen2.5-Math-7B model for RL, we find that the generated output of this base model is unstable, with useless and unverified code text, potentially due to its special pretraining data. Surprisingly, we find that the unexpected code snippets in the generated output do not significantly affect the RL performance in standard RL process. However, this meaningless code generation can be harmful for verification tasks. As a result, we add an additional code generation penalty of -0.5 for all generations containing code output. Additionally, we also add a short response penalty of -0.5 for the verification task when the model only outputs the final verification result without CoT. These rewards are specific to the Qwen 7B model. We believe that these auxiliary rewards have a minor effect on our main methodology since the model quickly convergess to these rewards in a few RL training steps.

Table 7: Hyperparameters for test-time scaling with self-verification and baselines

Parameter Category	**Self-Verification-Qwen-7B**	**Self-Verification-R1-1.5B**
Generation Settings		
Number of Samples	16 (unless specified)	16 (unless specified)
Temperature	0.6	0.6
Top-p	1.0	1.0
Max Tokens	2048	14336
Self-verification Configuration		
Verifier Max Tokens	2048	14336
Verifier Temperature	0.0	0.0
Verification Scale Coefficient α	1.0	0.1
Process-based Methods (Beam Search and DVTS) Configuration		
Beam Width	4, or 2 when number of samples is 2	4, or 2 when number of samples is 2
Step Delimiter	two linebreaks	two linebreaks
Max Search Steps	40	40



(a) AIME24 Score (avg@10)

(b) AIME25 Score (avg@10)

(c) AMC23 Score (avg@10)

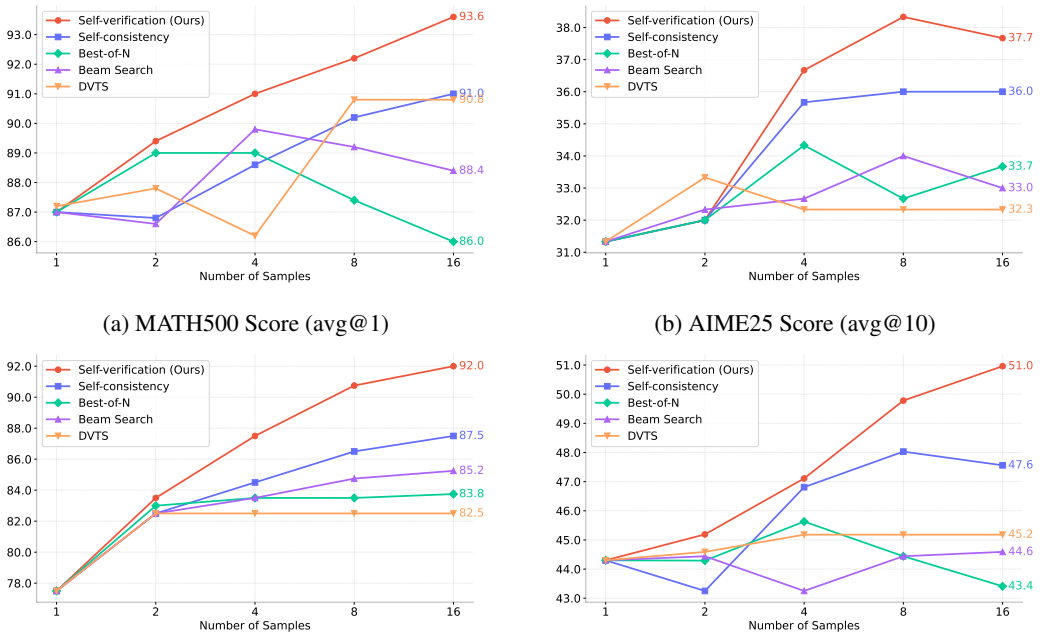
(d) OlympiadBench Score (avg@1)

Figure 5: Test-time scaling performance of Self-Verification-Qwen-7B on math reasoning benchmarks including AIME24, AIME25, AMC23, and OlympiadBench.

Online Buffer Management The online buffer is crucial for our self-verification framework. For Qwen-7B, we maintain a buffer size of 5000 samples, while for R1-1.5B, we expand it to 40000 samples to accommodate the larger dataset and longer responses. Both models update their buffers every 60 training steps (T_b) to ensure the verification data stays aligned with the current policy.

C Technical Details on Test-time Scaling

In this section, we provide the details of our test-time scaling process. We implement an efficient inference-time generation framework on top of Beeching et al. [53] but substitute the inference engine to SGLang [24], which provides a highly controllable frontend for guided generation. We use the



(a) MATH500 Score (avg@1)

(b) AIME25 Score (avg@10)

(c) AMC23 Score (avg@10)

(d) OlympiadBench Score (avg@1)

Figure 6: Test-time scaling performance of Self-Verification-R1-1.5B on math reasoning benchmarks including MATH500, AIME25, AMC23, and OlympiadBench.

same generation settings for all methods, whose hyperparameters are listed in Table 7. We choose the RLHFlow [46] PRM for all baselines as their reward models. According to the practice of Beeching et al. [53], we use the last step score of the PRM as the reward for the current step or response. On the hyperparameter α, which balances the answer selection between model verification score and the majority answer, we choose a relatively large value for Self-Verification-Qwen-7B model and a smaller value for Self-Verification-R1-1.5B. This is decided according to their verification accuracy during the training time, since we find the 7B model often has a better verification performance. In contrast, using smaller α value for the 1.5B model helps alleviate its verification error. For inference-time rollouts, we maintain the same temperature of 0.6 as that in the training time.

Computational Resources We typically use 8 NVIDIA GPUs with 80GB memory each for all test-time scaling experiments. We first deploy the SGLang server on all GPUs with data parallel strategy. For baselines requiring simultaneous reward model deployment, we lower the GPU memory utilization accordingly to enable additional memory for the reward model. The inference of reward models are directly through the Hugging Face Transformers library since it is non-trivial to deploy the PRM forward pass on our inference engine. In contrast, the verification process of our self-verification model can be fully realized by the LLM generation process. As a result, we only need to serve one model simply through the deployed SGLang server, where we show the efficiency of our framework at test time in Figure 4.

D Additional Results on Test-time Scaling

In Figure 5 and Figure 6, we show the test-time scaling performance of our Self-Verification-Qwen-7B and Self-Verification-R1-1.5B models on different math reasoning benchmarks, which corresponds to the results shown in the main paper. We find that our self-verification test-time method can achieve better performance than our compared baselines in most tasks.

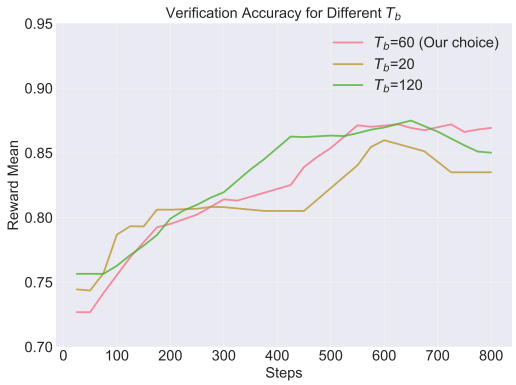


Figure 7: Verification accuracy on different choices of the policy-aligned buffer update frequency T_b from the Qwen2.5-Math-7B base model. The values are smoothed with a window size of 5.

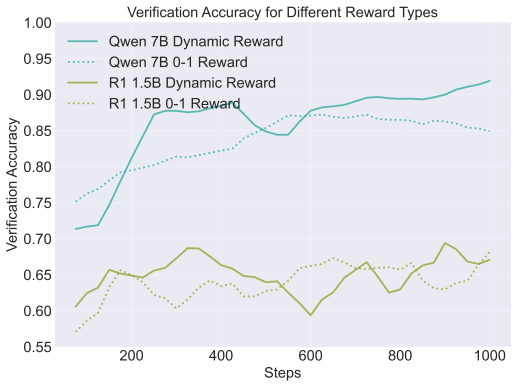


Figure 8: Verification accuracy difference between the dynamic reward design and a simple 0-1 reward for RL with Qwen 7B and R1 1.5B models. The values are smoothed with a window size of 5.

E Analysis on Distribution Discrepancy

To substantiate our claim that a distribution discrepancy exists between post-trained generators and general reward models (RMs), we evaluate the calibration of an external RM on outputs from different generators. A well-aligned RM should assign higher scores to correct answers regardless of the generator model. We measure the Pearson correlation coefficient between the RM scores and ground-truth correctness on the MATH500 dataset. We use the `RLHFlow/Llama3.1-8B-PRM-Deepseek-Data` model as the external RM. As shown in Table 8, the correlation drops significantly for responses generated by our post-trained model compared to standard instruction-tuned models, indicating that the external RM is misaligned with our model's output distribution. This misalignment limits the effectiveness of test-time scaling methods that rely on such RMs.

Table 8: Pearson correlation coefficient between external RM scores and ground-truth correctness on MATH500.

Generator Model	**Pearson Correlation Coefficient**
Llama-3.1-8B-Instruct	44.9
Qwen2.5-Math-7B-Instruct	42.3
Self-Verification-Qwen-7B	37.5

F Ablation Studies

We investigate how our framework's core design choices influence verification quality. First, we examine the policy-aligned buffer, which keeps the verifier on-policy by refreshing stored trajectories every T_b steps. As shown in Figure 7, the verification accuracy curves remain tightly clustered when sweeping T_b around our default value. While larger buffers amortize variance, they provide little extra signal during the online training process and thus show lower accuracy at later training stages. In contrast, $T_b = 20$ shows less stability during the training process. We therefore retain $T_b = 60$ as a robust trade-off.

Next, we analyze the dynamic verification reward, which reweights samples by their inferred difficulty. Figure 8 plots the difference in verification accuracy between using the dynamic reward and a simple binary reward. The figure shows that for the 7B model, the dynamic reward yields a significant and sustained positive accuracy difference, indicating a clear performance improvement. For the 1.5B model, the gain is less pronounced throughout training. This suggests that the smaller model's limited capacity may prevent it from fully leveraging the nuanced signal from the difficulty-aware reward.

The dynamic reward is efficient in our implementation since the difficulty statistics are derived from GRPO rollouts already generated during training.

G Comparison with Entropy-Minimization Baseline

Recent work has shown that entropy minimization (EM) can be a strong baseline for post-training LLMs. We compare our Self-Verification-Qwen-7B model with the one-shot entropy minimization model (EM 1-shot) from Gao et al. [56], which is also post-trained from the Qwen2.5-Math-7B base model. As shown in Table 9, our model outperforms the EM baseline across all benchmarks, demonstrating the effectiveness of our joint problem-solving and verification framework.

Table 9: Comparison with the entropy-minimization (EM) baseline on greedy-decoding scores.

Model	MATH500	AIME24 (avg@10)	AIME25 (avg@10)	AMC23 (avg@10)	Olympiad Bench
Self-Verification-Qwen-7B	83.6	20.00	16.67	63.75	34.81
EM 1-shot	68.8	19.33	8.67	54.50	27.26

H Prompting on Generative Verification

Our generative verification process follows the previous work [14]. Additionally, we adopt a multi-turn chat template to generate the verification output. A template is shown in the box below. We first append the problem and solution for the given data sample, and then add the verification prompt in a fixed format.

Listing 1: Generative Verification Prompting Template

```
User: [Math problem]

Assistant: [Solution]

User: Please verify the solution step by step. At the end of
   the solution verification, when you give your final grade,
   write it in the form "Is the answer correct (Yes/No)? X",
   where X is either Yes or No.

Assistant:
```

With this template, the LLM is prone to answer the verification question with 'Yes' or 'No' at the end. When performing generation with inference engine, we can simply set the stopping string at this template and thus we can acquire the probability of the 'Yes' token according to Equation (3).

I Examples of Problem-Solving and Verification Outputs

In this section, we provide some generation examples on problem-solving and verification tasks from our Self-Verification-Qwen-7B (Listing 2 and Listing 3) and Self-Verification-R1-1.5B models (Listing 4 and Listing 5).

Listing 2: Self-Verification-Qwen-7B Problem-Solving Example

```
User: Suppose that I have $6$ different books, $2$ of which are math books. In how
     many ways can I stack my $6$ books on a shelf if I do not want the math books to
      be next to each other?

Assistant: To determine the number of ways to stack 6 different books on a shelf such
      that the 2 math books are not next to each other, we can use the principle of
     complementary counting. This involves calculating the total number of possible
     arrangements of the books and then subtracting the number of arrangements where
     the math books are next to each other.
```

```
**Step 1: Calculate the total number of arrangements of the 6 books.**

The total number of ways to arrange 6 different books is given by the factorial of 6:
\[
6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720
\]

**Step 2: Calculate the number of arrangements where the 2 math books are next to
    each other.**

To treat the 2 math books as a single unit, we can think of them as forming a "super
    book." This reduces the problem to arranging 5 "books" (the "super book" and the
     other 4 individual books). The number of ways to arrange these 5 "books" is:
\[
5! = 5 \times 4 \times 3 \times 2 \times 1 = 120
\]
Within the "super book," the 2 math books can be arranged in 2 different ways (either
     math book 1 followed by math book 2, or math book 2 followed by math book 1).
    Therefore, the total number of arrangements where the 2 math books are next to
    each other is:
\[
5! \times 2 = 120 \times 2 = 240
\]

**Step 3: Subtract the number of arrangements where the math books are next to each
    other from the total number of arrangements.**

The number of arrangements where the 2 math books are not next to each other is:
\[
6! - 5! \times 2 = 720 - 240 = 480
\]

Thus, the number of ways to stack the 6 books on a shelf such that the math books are
     not next to each other is \(\boxed{480}\).
```

Listing 3: Self-Verification-Qwen-7B Verification Example

```
User: If $a$ is three times $b$, $b$ is twice $c$, and $c$ is four times $d$, what is
     the value of $\frac{a\cdot c}{b\cdot d}$?

Assistant: To find the value of \(\frac{a \cdot c}{b \cdot d}\), we start by
    expressing \(a\), \(b\), and \(c\) in terms of \(d\).

Given:
1. \(a = 3b\)
2. \(b = 2c\)
3. \(c = 4d\)

First, substitute \(c = 4d\) into \(b = 2c\):
\[b = 2(4d) = 8d.\]

Next, substitute \(b = 8d\) into \(a = 3b\):
\[a = 3(8d) = 24d.\]

Now we have \(a = 24d\), \(b = 8d\), and \(c = 4d\). We substitute these expressions
    into \(\frac{a \cdot c}{b \cdot d}\):
\[
\frac{a \cdot c}{b \cdot d} = \frac{(24d) \cdot (4d)}{(8d) \cdot d} = \frac{96d^2}{8d
    ^2} = \frac{96}{8} = 12.
\]

Therefore, the value is \(\boxed{12}\).

User: Please verify the solution step by step. At the end of the solution
    verification, when you give your final grade, write it in the form "Is the
    answer correct (Yes/No)? X", where X is either Yes or No.

Assistant: To verify the solution, let's re-evaluate the expression step by step.

Given:
1. \(a = 3b\)
2. \(b = 2c\)
3. \(c = 4d\)

First, express \(b\) in terms of \(d\):
\[b = 2c = 2(4d) = 8d.\]

Next, express \(a\) in terms of \(d\):
```

```
\[a = 3b = 3(8d) = 24d.\]

Now, substitute \(a = 24d\), \(b = 8d\), and \(c = 4d\) into the expression \(\frac{a
     \cdot c}{b \cdot d}\):
\[
\frac{a \cdot c}{b \cdot d} = \frac{(24d) \cdot (4d)}{(8d) \cdot d} = \frac{96d^2}{8d
     ^2} = \frac{96}{8} = 12.
\]

The value is indeed \(\boxed{12}\).

Is the answer correct (Yes/No)? Yes
```

Listing 4: Self-Verification-R1-1.5B Problem-solving Example

```
User: Suppose that I have $6$ different books, $2$ of which are math books. In how
     many ways can I stack my $6$ books on a shelf if I do not want the math books to
      be next to each other?

Assistant: (We omit the thinking part since it is too long.)

First, we calculate the total number of ways to arrange 6 different boosks without
     any restrictions. This is given by \(6!\):

\[
6! = 720
\]

Next, we need to subtract the number of arrangements where the two math books are
     next to each other. To find this, we treat the two math books as a single unit
     or "block." This reduces the problem to arranging 5 "books" (the "super book"
     and the other 4 individual books). The number of ways to arrange these 5 "books"
      is:
\[
5! = 5 \times 4 \times 3 \times 2 \times 1 = 120
\]
Within the "super book," the two math books can be arranged in 2 different ways (
     either math book 1 followed by math book 2, or math book 2 followed by math book
      1). Therefore, the total number of arrangements where the 2 math books are next
      to each other is:
\[
5! \times 2 = 120 \times 2 = 240
\]

Subtracting the number of bad arrangements from the total number of arrangements
     gives us the number of valid arrangements where the math books are not next to
     each other:

\[
720 - 240 = 480
\]

Thus, the number of ways to stack the 6 books on a shelf such that the math books are
      not next to each other is \(\boxed{480}\).
```

Listing 5: Self-Verification-R1-1.5B Verification Example

```
User: Given that events A and B are independent, and both are mutually exclusive with
      event C. It is known that $P(A) = 0.2$, $P(B) = 0.6$, and $P(C) = 0.14$. Find
     the probability that at least one of A, B, or C occurs, denoted as $P(A+B+C)$.

Assistant: Given that events A and B are independent, and both are mutually exclusive
      with event C, we need to find the probability that at least one of A, B, or C
     occurs, denoted as \( P(A \cup B \cup C) \).

We know the following probabilities:
- \( P(A) = 0.2 \)
- \( P(B) = 0.6 \)
- \( P(C) = 0.14 \)

Since A and B are independent, the probability of their intersection is:
\[ P(A \cap B) = P(A) \times P(B) = 0.2 \times 0.6 = 0.12 \]

Since A and B are mutually exclusive with C, the probabilities of A and C overlapping
     , and B and C overlapping, are both zero:
\[ P(A \cap C) = 0 \]
\[ P(B \cap C) = 0 \]
```

```
Using the inclusion-exclusion principle for three events, we have:
\[ P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C)
    + P(A \cap B \cap C) \]

Substituting the known values:
\[ P(A \cup B \cup C) = 0.2 + 0.6 + 0.14 - 0.12 - 0 - 0 + 0 \]
\[ P(A \cup B \cup C) = 0.2 + 0.6 + 0.14 - 0.12 \]
\[ P(A \cup B \cup C) = 0.94 - 0.12 \]
\[ P(A \cup B \cup C) = 0.82 \]

Thus, the probability that at least one of A, B, or C occurs is \(\boxed{0.82}\).

User: Please verify the solution step by step. At the end of the solution
    verification, when you give your final grade, write it in the form "Is the
    answer correct (Yes/No)? X", where X is either Yes or No.

Assistant:

<think>

Okay, so I need to verify if my answer of 0.82 is correct. Let me think through this
    step by step.

First, I recall that events A and B are independent. That means the probability of
    both A and B happening together is the product of their individual probabilities
    . So, P(A ∩ B) should be P(A) * P(B) = 0.2 * 0.6 = 0.12. That seems right.

Next, events A and B are mutually exclusive with C. That means A and C can't happen
    at the same time, and B and C can't happen at the same time. So, P(A ∩ C) = 0
    and P(B ∩ C) = 0. Okay, that makes sense.

Now, I used the inclusion-exclusion principle for three events to find P(A ∪ B ∪ C).
    The formula is:

P(A ∪ B ∪ C) = P(A) + P(B) + P(C) - P(A ∩ B) - P(A ∩ C) - P(B ∩ C) + P(A ∩ B ∩ C)

Plugging in the numbers:

P(A ∪ B ∪ C) = 0.2 + 0.6 + 0.14 - 0.12 - 0 - 0 + 0

Wait a minute, why are P(A ∩ C) and P(B ∩ C) zero? Because the events are mutually
    exclusive with C, so yes, they shouldn't overlap. But is there a chance that all
     three events A, B, and C might overlap? That is, is P(A ∩ B ∩ C) not zero?

Hmm, the problem says A and B are mutually exclusive with C, which I think means that
     A cannot occur with C, and B cannot occur with C. But A and B can happen
    separately from C.

So, perhaps the total probability is P(A) + P(B) + P(C) - P(A ∩ B) - P(A ∩ C) - P(B ∩
     C) + P(A ∩ B ∩ C). But since P(A ∩ C) is zero, and P(B ∩ C) is zero, and P(A ∩
    B) is 0.12, and P(A ∩ B ∩ C) is also zero,

Wait, maybe I'm confusing things.

If A and B are independent, then P(A ∩ B) = P(A) * P(B) = 0.12. Since A and B are
    mutually exclusive with C, that means P(A ∩ C) = 0 and P(B ∩ C) = 0. But what
    about P(A ∩ B ∩ C)? Since if A ∩ C is zero, and B ∩ C is zero, then A ∩ B ∩ C
    would also be zero because it's a subset of both A ∩ C and B ∩ C.

So, P(A ∩ B ∩ C) = 0.

Therefore, the formula simplifies to:

P(A ∪ B ∪ C) = P(A) + P(B) + P(C) - P(A ∩ B) - P(A ∩ C) - P(B ∩ C) + P(A ∩ B ∩ C)

Plugging in the numbers:

= 0.2 + 0.6 + 0.14 - 0.12 - 0 - 0 + 0

= 0.2 + 0.6 + 0.14 - 0.12

= 0.94 - 0.12

= 0.82

So, the calculation seems correct. Therefore, the answer of 0.82 is accurate.

</think>

Is the answer correct (Yes/No)? Yes
```

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: Yes. The claims made in the abstract and introduction are supported by the results of our paper in the experiments section.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Yes. The limitations of the work are discussed in Section 6.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [NA]

Justification: This work does not provide additional theoretical results. The theoretical correctness of this work follows the previous work on GRPO for LLM post-training.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: The technical details of our framework are disclosed in Appendix B and Appendix C. We also provide the code and data for reproducing the results in the supplementary material.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: The data and code are provided in the supplementary material.

Guidelines:

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

6. **Experimental setting/details**

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

Answer: [Yes]

Justification: The technical details on data, hyperparameters, and other information of our framework are disclosed in Appendix B and Appendix C.

Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [No]

Justification: Due to the extensive compute resources required to train our models through RL, we did not report error bars from repeated experiments associated with our method and experiments. An analysis of experimental stability is reported in Appendix B.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: The computational resources required for our experiments are stated in Appendix B and Appendix C.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The research conducted in the paper conforms with the NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: We discuss the broader impacts of our work in Appendix A.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [Yes]

    Justification: We discuss the safeguards of our work in Appendix A.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We credited the creators of the assets used in our paper and checked the licenses of the assets as stated in Appendix B and Appendix C.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [Yes]

    Justification: The code is contained in the supplementary material of this submission, which is well documented within the `readme` file.

    Guidelines:

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

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

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

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

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

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

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

    Answer: [NA]

    Justification: The methods in this paper do not use large language models (LLMs) in any non-standard way. LLMs were used for assistance with writing and editing.

    Guidelines:

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