Negative-Only Policy Optimization for One-Sided Verifiable Rewards

Jiacheng Xu[1,2] **Shuo He**[1] **Fuxiang Zhang**[1,2] **Chaojie Wang**[2] **Bo An**[1]
[1]Nanyang Technological University, Singapore [2]Skywork AI
jiacheng005@e.ntu.edu.sg, boan@ntu.edu.sg

Abstract

Reinforcement learning with verifiable rewards (RLVR) typically relies on a verifier that can reliably judge whether sampled solutions are correct. However, in many realistic settings, complete verification is generally unavailable, while cheap checks can only falsify failures. We formalize this regime as *one-sided verifiability* (OSV): verified negatives are reliable counterexamples, but verified positives are merely non-falsified candidates and may contain many false positives. This asymmetry makes standard RLVR unreliable, because reinforcing verified positives can directly reward shortcuts that pass partial checks. To address this issue, we propose *Negative-Only Policy Optimization* (NOPO), which treats verified positives as unlabeled and updates the policy only by suppressing trusted verified negatives. Specifically, NOPO combines a reference-relative negative suppression loss with a group-level adapter that scales each update by the proportion of non-falsified responses. Across unit-test, constraint-check, and LLM self-verifiers, NOPO improves pass@k over baselines on code, logic, and math benchmarks when oracle rewards are unavailable. These gains persist under inference-time scaling and cross-benchmark transfer.

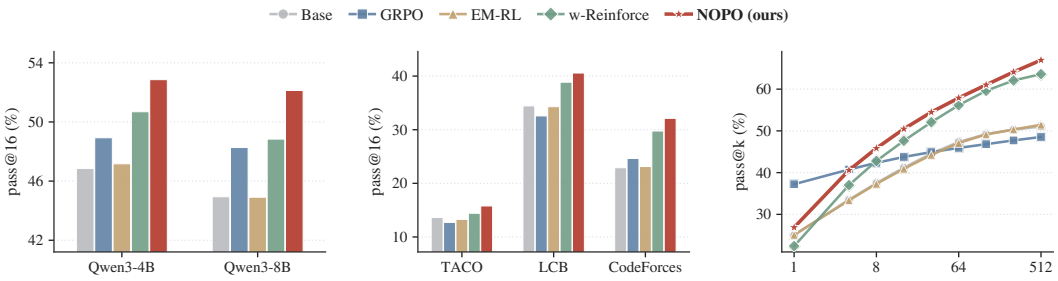


Figure 1: Overview of NOPO results. Left: average pass@16 across benchmarks from the main results in Table 2. Middle: cross-benchmark transfer of the Qwen3-4B model trained on CodeContests to other code benchmarks. Right: pass@k scaling on CodeContests for the Qwen3-4B model.

1 Introduction

Reinforcement learning with verifiable rewards (RLVR) has become a central recipe for improving reasoning-oriented LLMs, using a verifier to score sampled solutions and optimize the policy based on the resulting scores [18, 9, 28, 16]. This paradigm is most straightforward when the reward is complete: the verifier can reliably determine whether each sampled solution is correct. Outside curated training environments, however, constructing such complete verifiers becomes a bottleneck. Users may specify a task without being able to provide a full checker, and in open-ended scientific or reasoning domains, certifying a proposed answer can be as difficult as producing one. The central

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

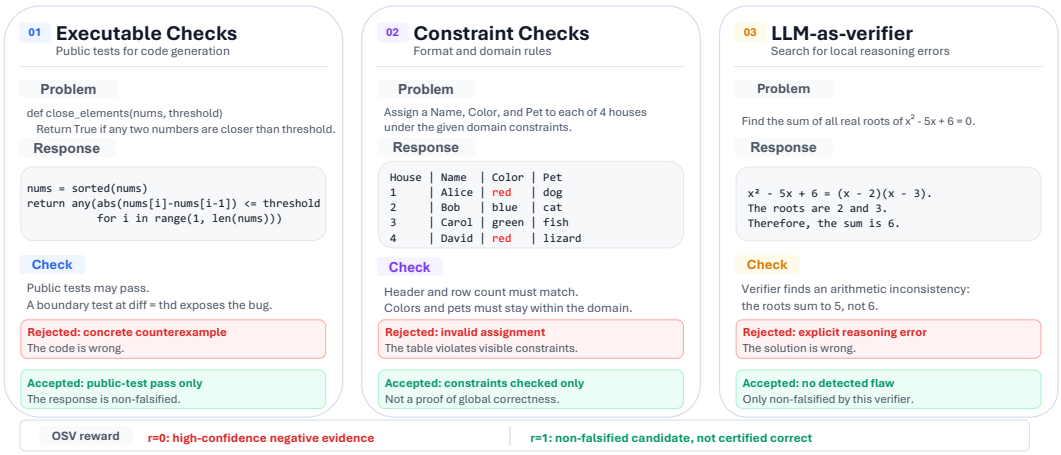


Figure 2: Illustration of one-sided verifiability across executable checks, constraint checks, and LLM-as-verifier settings. A rejection provides concrete evidence of failure, whereas acceptance only means that no checked violation was found; it does not certify correctness.

question is therefore not only how to learn from verifiable rewards, but how to learn when verification itself is incomplete.

The key observation of this paper is that incomplete verification is often asymmetric. Complete correctness may be unavailable, expensive, or as hard as solving the task, while local failures are much easier to detect. For example, in code generation, a failing unit test exposes a concrete bug even though a passing test suite does not establish full program correctness [3, 24, 37]. In structured reasoning tasks, explicit format or domain constraints can reject invalid answers without solving the task [23]. When no reliable task-specific checker is available, an LLM verifier may still identify obvious flaws in a proposed solution [4, 44, 40]. As illustrated in Figure 2, these checks provide one-sided evidence: a rejection is high-confidence evidence of failure, whereas an acceptance only means that no checked violation was found. We study this regime as *one-sided verifiability* (OSV): verifier-negative judgments have high precision, while verifier-positive judgments may contain a non-negligible false-positive component. Under OSV, a reward of 0 is high-confidence negative evidence, but a reward of 1 is not a trusted positive label and only a non-falsified candidate.

Existing approaches do not directly address this regime. Verifier-free or self-training methods construct surrogate learning signals from majority agreement, novelty, or confidence-like statistics [47, 45, 1, 36]. These methods show that useful signals can be obtained without ground-truth labels, but they do not exploit a common form of external feedback: reliable rejection without reliable acceptance. Conversely, standard RLVR methods such as GRPO [30] use the available verifier as a binary reward, reinforcing verifier-positive samples and suppressing verifier-negative samples. Under OSV, this binary interpretation is misleading. False positives are not merely noisy evaluations; once used as rewards, they become attractors of the policy optimization dynamics. Shortcut solutions that exploit unchecked failure modes receive direct positive gradients whenever they pass the partial verifier, allowing the policy to improve verifier pass rate without improving true correctness [33, 13].

In this paper, we propose *Negative-Only Policy Optimization* (NOPO), a policy optimization method designed for one-sided verifiable rewards. NOPO uses the verifier only for the evidence it can safely provide: high-confidence negatives. It suppresses verifier-negative trajectories while treating verifier-positive trajectories as unlabeled rather than correct. This negative-only principle creates a new optimization challenge: suppressing failures redistributes probability mass, but an all-failed rollout group provides no grounded direction for where that mass should move. NOPO addresses this challenge with two safeguards. At the sample level, it uses a reference-relative NPO loss whose gradient self-attenuates once a verifier-negative response is sufficiently downweighted relative to the reference policy. At the group level, it scales negative updates by the amount of non-falsified support in the same rollout group. Importantly, this support weight does not treat verifier-positive samples as correct demonstrations; it uses only their aggregate count to indicate whether the current policy has produced non-falsified alternatives toward which probability mass can be conservatively redistributed. Together, these components enable policy improvement from reliable negative evidence without directly reinforcing noisy positives.

Our main contributions are:

1. **One-sided verifiability.** We identify and formalize a common regime in realistic LLM tasks where local falsification is reliable but complete verification is unavailable. Under OSV, verifier negatives provide high-confidence evidence of failure, whereas verifier positives are only non-falsified candidates rather than trusted positive labels.
2. **Negative-Only Policy Optimization.** We propose NOPO, an RL method that learns from high-confidence negatives without directly reinforcing noisy positives. NOPO combines sample-level reference-relative NPO suppression with a group-level non-falsified support weight, yielding conservative policy improvement under one-sided verifiable rewards.
3. **Empirical validation.** We validate the OSV noise profile across multiple verifier types, showing that negative judgments are substantially cleaner than positive judgments. Under OSV, NOPO improves the model's capability frontier over standard RLVR baselines, and the gains persist under inference-time scaling and cross-benchmark transfer.

2 Preliminaries

Reinforcement Learning with Verifiable Rewards. Reinforcement Learning with Verifiable Rewards (RLVR) optimizes a policy π_θ using reward signals produced by a verifier. Given an input x and a sampled output $y \sim \pi_{\theta_{\text{old}}}(\cdot \mid x)$, the verifier returns a reward $r(x, y)$. A generic policy-gradient-style RLVR objective can be written as

$$\mathcal{J}_{\text{RLVR}}(\theta) = \mathop{\mathbb{E}}_{x\sim\mathcal{D}} \mathop{\mathbb{E}}_{y\sim\pi_{\theta_{\text{old}}}(\cdot|x)} \Big[\min\big(\rho(x,y)A(x,y),\, \bar{\rho}(x,y)A(x,y)\big) - \lambda_{\text{KL}}\, D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}}) \Big]. \quad (1)$$

Here $\rho(x, y) \coloneqq \frac{\pi_\theta(y|x)}{\pi_{\theta_{\text{old}}}(y|x)}$ is the importance ratio, $\bar{\rho}(x, y) \coloneqq \text{clip}(\rho(x, y), 1 - \epsilon, 1 + \epsilon)$ is its clipped version, $A(x, y)$ is an advantage term constructed from verifier rewards, and λ_{KL} controls KL regularization toward a reference policy π_{ref}. Different RLVR algorithms mainly differ in how they construct $A(x, y)$ from sampled rollouts and verifier signals.

GRPO [30] is one concrete RLVR instantiation and serves as a baseline in our experiments as well as the reference point for our later analysis. For each prompt x, GRPO samples a group of G outputs $\{y_i\}_{i=1}^G$ and defines

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

where r_i is the verifier reward of response y_i.

Negative Preference Optimization. LLM unlearning studies how to reduce a model's likelihood on a specified set of undesirable examples or behaviors while preserving its general capabilities. This is related to our setting in a limited but important sense: both require lowering the likelihood of trusted negative outputs without pushing the model arbitrarily far from its original behavior. Negative Preference Optimization (NPO) [41] is a representative unlearning objective for this purpose. For a negative response y to prompt x, define

$$\Delta_\theta(x, y) \coloneqq \log \pi_\theta(y \mid x) - \log \pi_{\text{ref}}(y \mid x).$$

The per-example NPO loss is

$$\mathcal{L}_{\text{NPO},\beta}(\theta; x, y) = \frac{2}{\beta} \log(1 + \exp(\beta\Delta_\theta(x, y))) \qquad \beta > 0. \quad (3)$$

A training objective averages this term over the trusted negative set. The inverse temperature β controls how sharply the loss transitions from penalizing likely negatives to attenuating already-suppressed negatives. The loss penalizes a response relative to its reference probability without requiring an explicit preferred response. Its derivative with respect to $\log \pi_\theta(y \mid x)$ is $2\sigma(\beta\Delta_\theta(x, y))$. Thus, the penalty is large when the current policy still assigns the negative response probability comparable to or above the reference policy, and becomes small once the response has already been suppressed below the reference.

Table 1: Pre-RL verifier noise across main benchmarks.

		Qwen3-4B				Qwen3-8B			
Benchmark	**Verifier**	pass@16	q_{neg}	q_{pos}	r_{neg}	pass@16	q_{neg}	q_{pos}	r_{neg}
TACO test	Public tests	13.62	100.0	64.5	87.9	16.12	100.0	66.8	82.5
LiveCodeBench	Public tests	34.44	100.0	24.0	89.2	34.86	100.0	25.9	88.0
CodeContests	Public tests	41.18	100.0	20.7	91.3	43.10	100.0	19.7	91.4
ZebraLogic	Constraint checks	71.25	100.0	73.0	34.8	56.25	100.0	77.5	56.2
OlympiadBench	LLM self-verifier	73.78	95.8	46.3	12.5	74.37	97.4	44.4	8.8

3 One-Sided Verifiability

3.1 Verifier Model and Examples

In the unsupervised RLVR setting [47, 10], the true correctness of sampled solutions is unobserved during training, so rewards must be constructed. Prior work derives rewards from model-internal or sample-level statistics, including agreement among sampled outputs [47] and verifier-free uncertainty objectives such as entropy minimization [1]. We study a different yet common supervision source: manually specified error-detecting checks. Such checks are often easy to define when full correctness is hard to verify, because they test necessary conditions rather than correctness itself. Failing a check provides evidence that a solution is wrong, whereas passing the available checks only indicates that no checked failure mode was detected.

Let x denote a problem instance and y a complete candidate solution. We distinguish between the latent ground-truth correctness

$$r^*(x, y) \in \{0, 1\},$$

and the observed reward

$$r(x, y) = V(x, y) \in \{0, 1\},$$

where V denotes a manually specified error-detection scheme available at training time. Such schemes are easy to obtain in many settings. We describe three common instances below.

Executable checks. In coding tasks, public unit tests are commonly used to check basic correctness properties of a solution. Given a set of input-output tests $\mathcal{T}_{\text{pub}}(x) = \{(I_j, O_j)\}_{j=1}^{m}$, these tests define a binary reward function:

$$V_{\text{pub}}(x, y) = \mathbb{I}[\forall (I_j, O_j) \in \mathcal{T}_{\text{pub}}(x),\ \text{Exec}(y, I_j) = O_j]\,. \tag{4}$$

Passing all public unit tests is only a necessary condition for correctness, not a sufficient one.

Constraint checks. For many realistic tasks, a correct solution must satisfy explicit constraints: structured outputs must follow the required format, assignments must respect domain restrictions, and intermediate claims must obey local consistency requirements. Let $\mathcal{C}(x) = \{c_j(x, \cdot)\}_{j=1}^{m_c}$ denote such necessary constraints induced by the task specification. These constraints define

$$V_{\text{con}}(x, y) = \mathbb{I}[\forall c_j \in \mathcal{C}(x),\ c_j(x, y) = 1]\,. \tag{5}$$

Failing a checked constraint is reliable evidence that the candidate is invalid, but satisfying all checked constraints does not certify that the full solution is correct.

LLM-as-verifier. For tasks with hard-to-formalize constraints, we use an LLM to search candidate solutions for errors rather than judge correctness, since certifying correctness is stronger than identifying a specific error. Let $J_\phi(x, y) \in \texttt{Find}, \texttt{NotFind}$ be its final binary judgment, where `Find` denotes an identified failure mode and `NotFind` denotes that none was found. We define the corresponding reward as

$$V_{\text{llm}}(x, y) = \mathbb{I}[J_\phi(x, y) = \texttt{NotFind}]\,. \tag{6}$$

Unlike executable or deterministic constraint checks, an LLM-based signal provides no structural protection against false negatives, as it may reject valid solutions. Thus, its one-sidedness must be empirically measured rather than assumed; Section 3.2 reports these measurements.

These examples instantiate the same broader problem. The observed reward $r(x, y)$ is an imperfect proxy for $r^*(x, y)$, but its errors are asymmetric: negative outcomes are much more reliable than

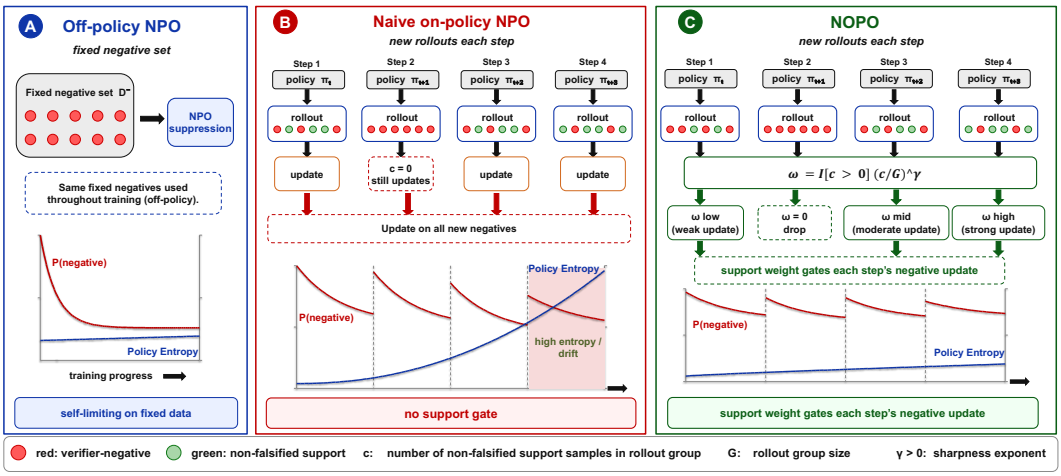


Figure 3: Overview of Negative-Only Policy Optimization (NOPO).

positive outcomes. The central question in this paper is not how to eliminate verifier noise completely, but how to perform RL when the available signal can reliably falsify some bad solutions but cannot reliably certify good ones.

3.2 Measuring One-Sidedness

We measure this asymmetry with two conditional quantities:

$$q_{\text{neg}} := \Pr(r^* = 0 \mid r = 0), \tag{7}$$
$$q_{\text{pos}} := \Pr(r^* = 0 \mid r = 1). \tag{8}$$

These quantities measure the probability that the ground-truth reward is false under each observed reward outcome: q_{neg} is the probability that a verifier-negative sample is truly incorrect, and q_{pos} is the probability that a verifier-positive sample is nevertheless incorrect. We call a verifier *one-sided verifiable* when q_{neg} is high but q_{pos} remains non-negligible. In this regime, observed negatives can be used as trusted counterexamples, whereas observed positives should be treated as non-falsified unlabeled candidates rather than ground-truth positives.

Table 1 reports q_{neg}, q_{pos}, and negative-class recall $r_{\text{neg}} := \Pr(r = 0 \mid r^* = 0)$, the fraction of truly incorrect responses flagged by the verifier. For public tests and constraint checks, $q_{\text{neg}} = 1$ by construction because they check necessary conditions for correctness. For the LLM self-verifier, q_{neg} is close to 1, showing accurate error identification with few rejected correct answers. Across all three verifier types, q_{pos} remains non-negligible, showing a substantial false-positive fraction among verifier-positive responses. Yet r_{neg} varies by an order of magnitude: code public tests flag 82–91% of true failures, ZebraLogic constraint checks 35–56%, and the LLM self-verifier on OlympiadBench only 9–13%. Thus, OSV asymmetry persists under different per-group negative densities. These quantities are computed using ground-truth labels r^* solely for analysis and unavailable during training.

Learning implications. This noise profile changes which verifier outcomes RLVR should trust. Standard RLVR treats $r = 1$ as a positive label and $r = 0$ as a negative label. Under OSV, however, the $r = 1$ pool mixes true solutions with unchecked failures. Reinforcing this pool gives shortcut trajectories a direct positive gradient whenever they survive the partial checks.

4 Negative-Only Policy Optimization

In the one-sided verifiable rewards (OSV) setting, the reliability of verifier-positive samples is generally unknown: q_{pos} is difficult to observe in realistic training environments. We therefore learn only from the reliable side of the signal, namely verifier-negative samples. This principle resembles Negative Preference Optimization (NPO), introduced in the preliminaries. NPO-style unlearning methods aim to reduce the likelihood of specified undesirable examples or behaviors relative to a reference policy, while preserving the model's original capabilities.

However, NPO-style algorithms are typically designed for off-policy updates on a fixed negative dataset, unlike on-policy RLVR. With fixed negatives, NPO suppresses undesirable examples relative to a reference policy and stops once their likelihood is sufficiently reduced. In RLVR, however, negatives are generated online by the changing policy itself. Negative-only updates therefore face a group-level ambiguity: suppressing rejected trajectories redistributes probability mass, but verifier-negative samples alone do not specify where it should move. Overly aggressive suppression can induce unconstrained "move-away" updates, causing the policy to generate new failures that are then further suppressed.

Directly applying NPO is therefore insufficient for OSV-based RL training. Although the NPO loss provides sample-level self-attenuation relative to the reference policy, it does not distinguish whether a verifier-negative rollout comes from a group that also contains non-falsified responses. This distinction is important in on-policy RLVR: when the same rollout group contains non-falsified responses, suppressing clear failures is more likely to redistribute probability mass toward the non-falsified part of the sampled group. To adapt NPO to this setting, we propose *Negative-Only Policy Optimization* (NOPO), which modulates NPO suppression by the amount of non-falsified support in the same rollout group. Importantly, verifier-positive samples are not treated as correct demonstrations. NOPO uses only their aggregate count as a support signal, indicating whether the current policy has produced non-falsified responses under the available verifier.

Non-falsified support weight. Given a prompt x, let $\mathbf{y} = (y_1, \ldots, y_G)$ be a rollout group sampled from the old policy $\pi_{\theta_{\text{old}}}(\cdot \mid x)$, and let $r_i = V(x, y_i) \in \{0, 1\}$ denote the verifier label of the i-th rollout. We define the amount of non-falsified support in the group as

$$c(\mathbf{r}) \coloneqq \sum_{i=1}^{G} r_i, \tag{9}$$

where $\mathbf{r} = (r_1, \ldots, r_G)$. If $c(\mathbf{r}) = 0$, the rollout group contains no non-falsified response under the available verifier, and negative-only suppression provides little information about a useful direction for probability redistribution. This event is common on hard prompts: under the rollout distribution, $\Pr[c(\mathbf{r}) = 0]$ equals one minus the verifier-pass@G rate, namely the probability that none of the G sampled responses passes the available verifier checks.

NOPO therefore weights each negative update by the non-falsified support within its rollout group:

$$\omega_\gamma(\mathbf{r}) \coloneqq \begin{cases} 1, & \gamma = 0, \\ \left(\dfrac{c(\mathbf{r})}{G}\right)^{\gamma}, & \gamma > 0, \end{cases} \qquad \gamma \geq 0. \tag{10}$$

The default $\gamma = 1$ gives a linear support weight. When $\gamma > 0$, groups with no verifier-positive responses receive zero weight automatically, low-support groups contribute less, and groups with more non-falsified support allow stronger suppression of verifier-negative responses. Setting $\gamma = 0$ recovers vanilla NPO, where all groups contribute uniformly, while larger γ makes NOPO more conservative on low-support groups.

The resulting NOPO objective applies NPO-style suppression only to verifier-negative responses, modulated by the group-level non-falsified support weight:

$$\mathcal{L}_{\text{NOPO}}(\theta) = \mathbb{E}_{x \sim \mathcal{D}, \mathbf{y} \sim \pi_{\theta_{\text{old}}}(\cdot|x)} \left[\omega_\gamma(\mathbf{r}) \cdot \frac{1}{G} \sum_{i=1}^{G} \mathbb{I}[r_i = 0] \, \mathcal{L}_{\text{NPO}}(\theta; x, y_i) \right], \tag{11}$$

where $r_i = V(x, y_i)$ and $\mathcal{L}_{\text{NPO}}$ is the reference-relative NPO loss with temperature β. The verifier labels and the support weight are treated as fixed feedback for the sampled rollout group; gradients are applied only through the NPO loss on verifier-negative responses. Algorithm 1 in Appendix B summarizes the corresponding training update.

This formulation separates reliability at two levels. The NPO term provides sample-level control: its gradient self-attenuates once the current policy assigns a negative response lower probability than the reference. The group weight $\omega_\gamma(\mathbf{r})$ provides group-level control: it trusts negative evidence only when the same on-policy rollout group contains verifier-positive support. As a result, NOPO updates on negative rollouts only when the group indicates that the prompt is within the model's current reach, avoiding unconstrained "move-away" updates on prompts where all sampled trajectories fail.

Table 2: Main results across code, logic, and math benchmarks under one-sided verifiable rewards.

Method	TACO	CodeContests	LCB	ZebraLogic	Olympiad	Avg.
Qwen3-4B						
Base	13.62	41.18	34.44	71.25	73.78	46.85
GRPO	15.89	43.75	36.17	74.38	74.44	48.93
EM-RL	15.44	40.91	35.26	69.37	74.89	47.17
w-Reinforce	18.62	47.63	38.52	75.00	73.70	50.69
NOPO	**19.52**	**50.47**	**39.61**	**79.37**	**75.33**	**52.86**
Qwen3-8B						
Base	16.12	43.10	34.86	56.25	74.37	44.94
GRPO	17.03	49.58	37.14	60.62	**76.97**	48.27
EM-RL	16.23	44.96	34.86	53.75	74.74	44.91
w-Reinforce	18.50	50.42	41.14	59.38	74.74	48.84
NOPO	**19.98**	**55.04**	**42.86**	**66.25**	76.52	**52.13**

5 Experiments

5.1 Setup

We use Qwen3-4B and Qwen3-8B as base models [38]. Code experiments cover LiveCodeBench [15], TACO test [20], and CodeContests [21] with public-test binary rewards, with CodeForces reserved as an unseen cross-benchmark transfer target. Logic experiments use ZebraLogic with a deterministic constraint checker that verifies format and domain constraints without certifying full correctness, while math experiments use OlympiadBench [11] with an LLM self-verifier binary reward. We compare four methods, each using $G = 16$ samples per prompt: **GRPO**, which directly optimizes the observed verifier reward; **EM-RL**, a token-level verifier-free baseline that minimizes policy entropy during training [1]; **w-Reinforce**, which assigns verifier-negative samples greater weight than positive ones [46]; and **NOPO**, our proposed method.

5.2 Main Result

As shown in Table 2, NOPO achieves the best average pass@16 on both backbones among RL methods, with w-Reinforce second on average. Both methods treat verifier-negative samples as trustworthy supervision but differ in how they control policy drift from negative-only penalties: w-Reinforce gives verifier-positive samples a small positive weight to offset the dominant negative gradient, whereas NOPO directly rescales each prompt's negative-only loss by its within-group non-falsified support. By contrast, verifier-free EM-RL, which uses only token-level entropy minimization, gives essentially no gain over Base on either backbone, consistent with the view that intrinsic-reward methods are bounded by sharpening the base policy's initial distribution [1, 43, 39, 10]. At the per-cell level, NOPO ranks first in 9 of 10 benchmark–backbone cells; the only exception is OlympiadBench under Qwen3-8B, where it ranks second behind GRPO. As Table 1 shows, OlympiadBench has the lowest verifier negative-class recall in our setup, so few truly incorrect rollouts are flagged, weakening NOPO's negative-only supervision signal.

5.3 Inference-Time Scaling

A central debate around RLVR is whether it expands a model's capability frontier [25] or merely concentrates probability mass on solutions already discoverable at large k [39, 45]. We therefore test whether our gains persist at large inference budgets, rather than reflecting sharpening around solutions reachable by the base model with enough samples. Table 3 reports Qwen3-4B on CodeContests and LiveCodeBench as k scales from 1 to 512. NOPO shows an advantage at larger inference budgets, and this advantage *widens* with k, indicating genuine frontier expansion: the post-RL policy solves problems the base model cannot reach even with hundreds of attempts. w-Reinforce, which also suppresses verifier-negative outputs, is the consistent runner-up at large k, supporting the OSV view that trusted-negative suppression drives frontier expansion. NOPO's per-prompt non-falsified support weight further improves on this principle. GRPO, by contrast, illustrates the sharpening failure mode: its pass@1 is the highest across methods, but at $k = 512$ it falls *below* the base model on both benchmarks, indicating that small-k gains come at the cost of exploration. More encouragingly,

Table 3: Inference-time pass@k on Qwen3-4B. **Bold** marks the maximum, underline the second-best.

Method	$k=1$	$k=4$	$k=8$	$k=16$	$k=32$	$k=64$	$k=128$	$k=256$	$k=512$
CodeContests									
Base	25.04	33.51	37.50	41.18	44.55	47.35	49.21	50.29	51.05
GRPO	**37.23**	**40.74**	42.35	43.75	44.92	45.92	46.82	47.72	48.54
EM-RL	25.08	33.36	37.30	40.91	44.23	47.09	49.15	50.34	51.46
w-Reinforce	22.39	37.01	42.80	47.63	52.08	56.16	59.61	62.08	63.60
NOPO (ours)	26.87	40.58	**45.84**	**50.47**	**54.48**	**57.89**	**61.03**	**64.11**	**66.95**
LiveCodeBench									
Base	26.03	31.11	32.89	34.44	35.93	37.30	38.58	40.07	42.29
GRPO	**29.40**	33.07	34.58	36.17	37.66	38.96	40.13	41.03	41.71
EM-RL	25.74	31.41	33.39	35.26	37.04	38.61	40.05	41.56	43.43
w-Reinforce	26.21	33.45	36.21	38.52	40.55	42.64	44.74	46.75	48.57
NOPO (ours)	25.33	**33.72**	**36.75**	**39.61**	**42.34**	**44.89**	**47.04**	**48.88**	**50.86**

Table 4: Cross-bench transfer pass@16 on Qwen3-4B with two training sources. **Bold** marks the row maximum and underline the second-best among RL methods.

Source	**Target**	**Base**	**GRPO**	**EM-RL**	**w-Reinforce**	**NOPO (ours)**
CodeContests	TACO	13.62	12.71	13.28	14.42	**15.78**
	LiveCodeBench	34.44	32.57	34.29	38.86	**40.57**
	CodeForces	22.91	24.63	23.13	29.76	**32.12**
	Avg-Unseen	23.66	23.30	23.57	27.68	**29.49**
LiveCodeBench	TACO	13.62	12.49	13.62	15.55	**16.00**
	CodeContests	41.18	39.75	43.10	47.70	**49.37**
	CodeForces	22.91	22.48	24.84	29.34	**30.41**
	Avg-Unseen	25.90	24.91	27.19	30.86	**31.93**

NOPO's frontier-expanding behavior suggests that modulating negative-only updates by per-prompt non-falsified support may transfer to other RLVR settings as a remedy for entropy collapse.

5.4 Policy generalization

To test whether OSV-trained gains transfer beyond the source benchmark, Table 4 evaluates checkpoints trained on CodeContests and LiveCodeBench on unseen code benchmarks. GRPO again shows limited generalization: its main-result gains fail to transfer even within the same domain, suggesting source-benchmark overfitting rather than genuine capability improvement. By contrast, w-Reinforce and NOPO, both focused on verifier-negative samples, retain clear transfer gains, and NOPO is best on every unseen target. This highlights the advantage of effectively leveraging negative samples under OSV. EM-RL, lacking a sustained effective optimization signal, performs on par with Base.

5.5 Ablation: support exponent γ and naive baselines

This ablation tests the necessity of the non-falsified support weight and its robustness to the support exponent γ. Here, *NPO* directly applies the NPO loss in RL, while *NSR* is w-Reinforce without verifier-positive optimization. We compare $\gamma \in \{0.5, 1.0, 2.0\}$, with $\gamma = 1.0$ as the default. All variants are trained on TACO with Qwen3-4B and evaluated on three unseen code targets. Non-falsified support weights outperform both NSR and NPO in-domain and under cross-benchmark transfer, confirming their necessity. The linear support weight ($\gamma = 1.0$) performs best among the three exponents. Detailed transfer results are in Table 6.

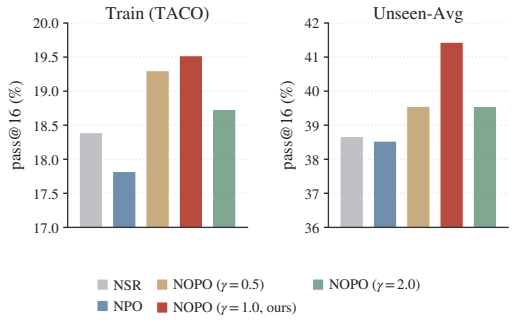


Figure 4: Ablation study on Qwen3-4B trained on TACO and evaluated on unseen benchmarks

6 Related Work

Unsupervised RLVR. Unsupervised RLVR can be categorized into two regimes by reward source [10]. The *intrinsic* regime derives rewards from the model itself through certainty- or consistency-based signals [47, 22, 1, 45, 43, 12], and is provably bounded by the sharpening of the policy's initial distribution. The *external* regime grounds rewards in mechanisms outside the model and typically relies on verifiers that are effectively sound and complete, such as formal proof checkers, deterministic execution against complete test suites, or exact symbolic checks [14, 42, 32]. Our setting falls within this external regime but relaxes its soundness-and-completeness assumption: readily available external verifiers in realistic deployment settings often cannot certify correctness, but can reliably reject visible failures, an asymmetry we formalize as one-sided verifiability.

RLVR with Noisy Rewards. A growing line of work studies RLVR when the verifier is imperfect [29, 2, 5], and existing methods share a common recipe: characterize the verifier's noise—either through an aggregate quality index or through separately modeled false-positive and false-negative rates—and plug this characterization into the policy gradient as a correction or weighting term. Obtaining such a characterization itself requires ground-truth labels for calibration, which our setting explicitly disallows; moreover, the rate at which the verifier falsely accepts incorrect outputs drifts upward as the policy learns to satisfy partial checks, so calibrated pre-training estimates or online noise-rate estimates may be unreliable in the OSV regime. NOPO instead works structurally rather than statistically: it neither estimates nor corrects for noise rates, using the verifier only for the high-confidence negatives it can safely provide, consistent with the broader observation that false-positive rewards are particularly harmful as they reinforce shortcut behaviors [13].

Negative-Oriented Optimization. NOPO is also related to negative-oriented optimization. LLM unlearning removes undesirable knowledge or behaviors while preserving general utility [6, 26, 19]. Negative preference objectives such as NPO suppress a fixed off-policy forget set while avoiding naive gradient-ascent instability, with later variants revisiting the reference model [41, 7]. NOPO adopts NPO's self-attenuating suppression principle but applies it to on-policy verifier-negative rollouts, which introduces a redistribution ambiguity: negatives specify what to avoid, not where probability mass should move. This motivates our group-level non-falsified support gate. Another related line, negative sample reinforcement in RLVR, shows that penalizing incorrect rollouts can improve reasoning and preserve diversity [46, 34, 27, 8]. Unlike these methods, NOPO targets OSV, where verifier positives are unlabeled non-falsified support rather than reliable positive rewards.

7 Conclusion

As LLM capabilities improve and their applications expand, constructing complete verifiers for increasingly diverse training data becomes increasingly costly. This paper studies a more practical setting where only partial verification is available. We show that verifier rewards in this setting often exhibit one-sided verifiability: rejected samples provide reliable evidence of failure, whereas accepted samples are merely non-falsified rather than certified correct. Under this asymmetry, directly treating verifier-positive samples as positive rewards introduces substantial noise and may encourage reward hacking rather than expand the model's true capability frontier. We therefore propose Negative-Only Policy Optimization (NOPO), which learns only from high-confidence negative evidence. NOPO combines sample-level reference-relative NPO suppression with group-level non-falsified support weighting, enabling controlled negative updates without directly reinforcing noisy positives. Empirically, NOPO improves performance across multiple verifier types, with advantages that persist under inference-time scaling and cross-benchmark transfer. These results suggest that properly leveraging negative evidence can substantially expand the model's capability frontier without ground-truth verifiers, making negative-only optimization a promising direction for scalable RLVR.

Limitations. Negative-only learning is inherently delicate: it specifies what the policy should avoid but not where removed probability mass should move. This limitation motivates NOPO's conservative design and suggests future extensions. In different one-sided verifiability scenarios, verifier-positive samples may contain exploitable structure, such as pairwise comparisons or uncertainty-based filtering, that could yield partial positive signals. Incorporating such signals while preserving the robustness of negative-only optimization remains an important direction for future work.

Acknowledgments and Disclosure of Funding

Nil.

References

[1] Shivam Agarwal, Zimin Zhang, Lifan Yuan, Jiawei Han, and Hao Peng. The unreasonable effectiveness of entropy minimization in LLM reasoning. *arXiv preprint arXiv:2505.15134*, 2025.

[2] Xin-Qiang Cai, Wei Wang, Feng Liu, Tongliang Liu, Gang Niu, and Masashi Sugiyama. Reinforcement learning with verifiable yet noisy rewards under imperfect verifiers. *arXiv preprint arXiv:2510.00915*, 2025.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[4] 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. *arXiv preprint arXiv:2110.14168*, 2021.

[5] Omar El Mansouri, Mohamed El Amine Seddik, and Salem Lahlou. Noise-corrected GRPO: From noisy rewards to unbiased gradients. *arXiv preprint arXiv:2510.18924*, 2025.

[6] Ronen Eldan and Mark Russinovich. Who's Harry Potter? approximate unlearning in LLMs. *arXiv preprint arXiv:2310.02238*, 2023.

[7] Chongyu Fan, Jiancheng Liu, Licong Lin, Jinghan Jia, Ruiqi Zhang, Song Mei, and Sijia Liu. Simplicity prevails: Rethinking negative preference optimization for LLM unlearning. In *Advances in Neural Information Processing Systems (NeurIPS)*, San Diego, CA, 2025.

[8] Yunzhen Feng, Parag Jain, Anthony Hartshorn, Yaqi Duan, and Julia Kempe. Don't waste mistakes: Leveraging negative RL-groups via confidence reweighting. *arXiv preprint arXiv:2510.08696*, 2025.

[9] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[10] Bingxiang He, Yuxin Zuo, Zeyuan Liu, Shangziqi Zhao, Zixuan Fu, Junlin Yang, Cheng Qian, Kaiyan Zhang, Yuchen Fan, Ganqu Cui, Xiusi Chen, Youbang Sun, Xingtai Lv, Xuekai Zhu, Li Sheng, Ran Li, Huan-Ang Gao, Yuchen Zhang, Bowen Zhou, Zhiyuan Liu, and Ning Ding. How far can unsupervised RLVR scale LLM training? In *International Conference on Learning Representations (ICLR)*, 2026.

[11] 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 *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3828–3850, Bangkok, Thailand, 2024.

[12] Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-Zero: Self-evolving reasoning LLM from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

[13] Sukai Huang, Shu-Wei Liu, Nir Lipovetzky, and Trevor Cohn. The dark side of rich rewards: Understanding and mitigating noise in VLM rewards. In *PRL Workshop @ International Conference on Automated Planning and Scheduling (ICAPS)*, 2025.

[14] Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651:607–613, 2025.

[15] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations (ICLR)*, 2025.

[16] Kimi Team. Kimi k1.5: Scaling reinforcement learning with LLMs. *arXiv preprint arXiv:2501.12599*, 2025.

[17] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Symposium on Operating Systems Principles (SOSP)*, Koblenz, Germany, 2023.

[18] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

[19] Nathaniel Li, Alexander Pan, Anjali Gopal, Summer Yue, Daniel Berrios, Alice Gatti, Justin D. Li, Ann-Kathrin Dombrowski, Shashwat Goel, Long Phan, Gabriel Mukobi, Nathan Helm-Burger, Rassin Lababidi, Lennart Justen, Andrew B. Liu, Michael Chen, Isabelle Barrass, Oliver Zhang, Xiaoyuan Zhu, Rishub Tamirisa, Bhrugu Bharathi, Adam Khoja, Zhenqi Zhao, Ariel Herbert-Voss, Cort B. Breuer, Samuel Marks, Oam Patel, Andy Zou, Mantas Mazeika, Zifan Wang, Palash Oswal, Weiran Lin, Adam A. Hunt, Justin Tienken-Harder, Kevin Y. Shih, Kemper Talley, John Guan, Russell Kaplan, Ian Steneker, David Campbell, Brad Jokubaitis, Alex Levinson, Jean Wang, William Qian, Kallol Krishna Karmakar, Steven Basart, Stephen Fitz, Mindy Levine, Ponnurangam Kumaraguru, Uday Tupakula, Vijay Varadharajan, Ruoyu Wang, Yan Shoshitaishvili, Jimmy Ba, Kevin M. Esvelt, Alexandr Wang, and Dan Hendrycks. The WMDP benchmark: Measuring and reducing malicious use with unlearning. In *International Conference on Machine Learning (ICML)*, Vienna, 2024.

[20] Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. TACO: Topics in algorithmic COde generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

[21] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022.

[22] Ruotong Liao, Nikolai Röhrich, Xiaohan Wang, Yuhui Zhang, Yasaman Samadzadeh, Volker Tresp, and Serena Yeung-Levy. Tool verification for test-time reinforcement learning. *arXiv preprint arXiv:2603.02203*, 2026.

[23] Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. ZebraLogic: On the scaling limits of LLMs for logical reasoning. In *International Conference on Machine Learning (ICML)*, 2025.

[24] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, New Orleans, LA, 2023.

[25] Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. ProRL: Prolonged reinforcement learning expands reasoning boundaries in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, San Diego, CA, 2025.

[26] Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary C. Lipton, and J. Zico Kolter. TOFU: A task of fictitious unlearning for LLMs. In *Conference on Language Modeling (COLM)*, Philadelphia, PA, 2024.

[27] Gongrui Nan, Siye Chen, Jing Huang, Mengyu Lu, Dexun Wang, Chunmei Xie, Weiqi Xiong, Xianzhou Zeng, Qixuan Zhou, Yadong Li, and Xingzhong Xu. NGRPO: Negative-enhanced group relative policy optimization. *arXiv preprint arXiv:2509.18851*, 2025.

[28] OpenAI. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

[29] Ali Rad, Khashayar Filom, Darioush Keivan, Peyman Mohajerin Esfahani, and Ehsan Kamalinejad. Rate or fate? RLVeR: Reinforcement learning with verifiable noisy rewards. *arXiv preprint arXiv:2601.04411*, 2026.

[30] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[31] 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 (EuroSys)*, pages 1279–1297, Rotterdam, Netherlands, 2025.

[32] Toby Simonds and Akira Yoshiyama. LADDER: Self-improving LLMs through recursive problem decomposition. *arXiv preprint arXiv:2503.00735*, 2025.

[33] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems (NeurIPS)*, New Orleans, LA, 2022.

[34] Xinyu Tang, Yuliang Zhan, Zhixun Li, Wayne Xin Zhao, Zhenduo Zhang, Zujie Wen, Zhiqiang Zhang, and Jun Zhou. Rethinking sample polarity in reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2512.21625*, 2025.

[35] Yinjie Wang, Ling Yang, Ye Tian, Ke Shen, and Mengdi Wang. Co-evolving LLM coder and unit tester via reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, San Diego, CA, 2025. Spotlight.

[36] Jiacheng Xu, Feng Chen, Xiuneng Xu, and Bo An. Entropy-regularized rank-masked policy optimization for test-time reinforcement learning in code generation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 47843–47857, Budapest, Hungary, 2026.

[37] Jiacheng Xu, Wentao Zhang, Zhiyi Lyu, Fuxiang Zhang, Chaojie Wang, Yang Liu, and Bo An. Two-stage reinforcement learning for sound and adversarial test generation in code LLMs. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 37031–37051, Budapest, Hungary, 2026.

[38] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[39] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems (NeurIPS)*, San Diego, CA, 2025. Oral.

[40] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *International Conference on Learning Representations (ICLR)*, Singapore, 2025.

[41] Ruiqi Zhang, Licong Lin, Yu Bai, and Song Mei. Negative preference optimization: From catastrophic collapse to effective unlearning. In *Conference on Language Modeling (COLM)*, Philadelphia, PA, 2024.

[42] Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. In *Advances in Neural Information Processing Systems (NeurIPS)*, San Diego, CA, 2025. Spotlight.

[43] Xuandong Zhao, Zhewei Kang, Aosong Feng, Sergey Levine, and Dawn Song. Learning to reason without external rewards. In *International Conference on Learning Representations (ICLR)*, 2026.

[44] 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 (NeurIPS)*, New Orleans, LA, 2023.

[45] Yujun Zhou, Zhenwen Liang, Haolin Liu, Wenhao Yu, Kishan Panaganti, Linfeng Song, Dian Yu, Xiangliang Zhang, Haitao Mi, and Dong Yu. Evolving language models without labels: Majority drives selection, novelty promotes variation. *arXiv preprint arXiv:2509.15194*, 2025.

[46] Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. The surprising effectiveness of negative reinforcement in LLM reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, San Diego, CA, 2025.

[47] Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. TTRL: Test-time reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, San Diego, CA, 2025.

A Broader impacts.

A practical benefit of OSV and NOPO is that they can reduce the need for expensive complete verifiers, making RLVR-style training more accessible in domains where only partial checks are available. At the same time, stronger reasoning and code-generation checkpoints can be misused, and optimization against incomplete checks can amplify unchecked failure modes if deployed without careful evaluation.

B NOPO training update

Algorithm 1 gives the training update used for Negative-Only Policy Optimization.

Algorithm 1 NOPO training update.

Input: dataset $\mathcal{D}$, policy π_θ, reference policy π_{ref}, verifier V, group size G, support exponent γ, NPO temperature β
for each training step **do**
  Sample prompt $x \sim \mathcal{D}$
  Set $\pi_{\theta_{\mathrm{old}}} \leftarrow \pi_\theta$ and generate rollouts $\mathbf{y} = (y_1, \ldots, y_G)$ with $y_i \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid x)$
  Compute verifier labels $r_i = V(x, y_i)$ for $i = 1, \ldots, G$
  Compute $c(\mathbf{r}) = \sum_{i=1}^{G} r_i$ and $\omega_\gamma(\mathbf{r})$ by Eq. 10
  Take a gradient step on $\mathcal{L}_{\mathrm{NOPO}}(\theta)$ in Eq. 11
end for
Output: trained policy π_θ

C Experimental Details

C.1 Benchmark

All datasets involved are publicly available. For LiveCodeBench, we use the code-generation split of `release_v6`. For TACO, we use the evaluation set released by TCS [37] and exclude 64 problems for which our parser cannot extract public test cases. For CodeContests, we use the evaluation set released by CURE [35]. For ZebraLogic, we use the Medium-Large subset (4×4, 4×5, 5×3, and 6×2). Unless otherwise specified above, datasets are used as provided. For the main results in Table 2, training and evaluation use the same test set for each benchmark, whereas the transfer experiments in Table 4 evaluate on target benchmarks not used for training. Only one-sided verifier rewards are available during training, and all methods are evaluated using the oracle. Table 5 summarizes the training verifiers and evaluation oracles.

Table 5: Training rewards, evaluation oracles, and mean test counts per problem.

Benchmark	Training reward	Evaluation oracle	Tests: OSV / oracle
TACO	public-test binary reward	full test suite (public + hidden)	1.7 / 204
CodeContests	public-test binary reward	full test suite (public + hidden)	1.9 / 12
LCB	public-test binary reward	full test suite (public + hidden)	2.6 / 40
ZebraLogic	constraint checker	ground-truth answer	–
OlympiadBench	LLM self-verifier	ground-truth answer	–

C.2 Experimental Configuration

All RL runs use the verl framework [31] with vLLM rollouts [17] and an FSDP actor. The two backbones are Qwen3-4B and Qwen3-8B in non-thinking mode. Unless noted otherwise, every method shares the optimisation, sampling, and bookkeeping settings below; only the loss term and its method-specific hyperparameter change between methods.

Optimisation. We use AdamW with a constant learning rate of 5×10^{-7} after a 20-step linear warmup. The training batch is 16 prompts; each prompt produces $G = 16$ on-policy rollouts, giving a $16 \times 16 = 256$-rollout step. PPO mini-batches contain 4 prompts and we run a single inner PPO epoch per global step. The maximum number of global steps is 150. For all methods dominated by negative optimisation, prolonged training may lead to entropy divergence, so we monitor policy entropy during training and retain the most recent checkpoint saved before an observed entropy divergence for evaluation.

Sampling. A single sampling profile is shared between RL rollouts and validation rollouts. For Qwen3 models in non-thinking mode, generation uses `temperature`=0.8, `top_p`=0.8, `top_k`=20, and `min_p`=0.

Sequence lengths. Code experiments use `max_response_length`=8192. Logic and math experiments use `max_response_length`=16384.

Method-specific hyperparameters.

- **GRPO**: kl_loss_coef=10^{-3}, no other deviations from the verl default.
- **w-Reinforce** [46]: verifier-positive samples receive weight $\lambda = 0.1$, verifier-negative samples receive weight 1.
- **EM-RL** [1]: `adv_estimator=rloo`; the training reward is replaced by token-level entropy minimisation computed inside the trainer from actor log-probabilities, and the environment reward manager returns zero placeholders on the training path. Validation still routes through the task-native evaluator so pass@k is reported on the same scale as the other methods.
- **NPO**: $\beta = 0.1$, `kl_loss_coef`=0. The reference policy π_{ref} is the frozen base model.
- **NOPO** (default): $\beta = 0.1$, $\gamma = 1.0$, `kl_loss_coef`=0. The non-falsified support weight $\omega_\gamma(\mathbf{r})$ from Eq. 10 is computed from the same on-policy rollout group used for the NPO term; $c(\mathbf{r})$=0 groups receive zero weight automatically and are dropped from the actor update.

Inference-time scaling. For the inference-scaling sweep (Table 3), we resample $n = 512$ rollouts per problem from each method's checkpoint with the same sampling profile and report pass@k as a function of $k \in \{1, 4, 8, 16, 32, 64, 128, 256, 512\}$.

Compute resources. All RL training runs were executed on eight NVIDIA A100 GPUs with 80GB memory per GPU. A typical (method, backbone, benchmark) run takes approximately 12 hours under this setup. Evaluation-only sweeps reuse selected checkpoints and are cheaper than training, although the inference-time scaling results in Table 3 require generating up to 512 rollouts per problem and still take approximately 12 hours under the same configuration.

D Per-target ablation numbers

Table 6 expands Figure 4 with per-target pass@16 for the five-method ablation on Qwen3-4B trained on TACO. **Train** is in-domain pass@16; the remaining columns are unseen-transfer pass@16 ($G = 16$ inference samples per problem) for the same selected checkpoint. **Unseen-Avg** is the macro mean of the three unseen-transfer columns and matches the right-hand bars of Figure 4. **Bold** marks the column maximum.

Table 6: Per-target pass@16 for the Section 5.5 ablation on Qwen3-4B trained on TACO.

Method	Train	LiveCodeBench	CodeContests	CodeForces	Unseen-Avg
NSR	18.39	37.71	48.95	29.34	38.67
NPO	17.82	37.71	47.70	30.19	38.53
NOPO ($\gamma = 0.5$)	19.30	38.86	48.54	31.26	39.55
NOPO ($\gamma = 1.0$, default)	**19.52**	**39.43**	**51.05**	**33.83**	**41.44**
NOPO ($\gamma = 2.0$)	18.73	38.86	48.95	30.84	39.55

E Post-training verifier noise

Table 7 reports post-RL verifier noise for Qwen3-4B trained on CodeContests, with Base statistics from Table 1 and unseen-target averages from Table 4. Public tests check necessary conditions for correctness, so q_{neg} remains 100% across all policies. GRPO increases q_{pos} and decreases r_{neg}, indicating more verifier-escaping errors. In contrast, w-Reinforce and NOPO reduce q_{pos} while maintaining high r_{neg}, with NOPO achieving the lowest q_{pos} and the highest in-domain and average transfer performance.

Table 7: Post-RL verifier noise and oracle performance on CodeContests with Qwen3-4B. All values are percentages.

Policy	q_{neg}	q_{pos}	r_{neg}	In-domain pass@16	Transfer average
Base	100.0	20.7	91.3	41.18	23.66
GRPO	100.0	25.2	79.8	43.75	23.30
w-Reinforce	100.0	18.1	93.4	47.63	27.68
NOPO	100.0	16.6	92.4	50.47	29.49

F Per-problem overlap at a fixed inference budget

We define the capability frontier operationally as the set of problems solved under a fixed inference budget. Following the Qwen3-4B scaling experiment in Table 3, a problem is counted as solved if at least one of its 512 oracle-evaluated samples is correct. Table 8 shows that NOPO retains $120/122$ Base-solved CodeContests problems and $73/74$ Base-solved LiveCodeBench problems, while additionally solving 40 and 16 problems, respectively. Thus, NOPO nearly preserves the Base-solved set while enlarging the reachable problem set under the same sampling budget.

Table 8: Problem-level solution overlap at 512 samples per problem, Qwen3-4B.

Benchmark	Total	Base solved	NOPO solved	Both	Neither	Base-only	NOPO-only
CodeContests	239	122	160	120	77	2	40
LiveCodeBench	175	74	89	73	85	1	16

G Verifier characteristics and NOPO gains

Table 9 compares verifier asymmetry with NOPO's pass@16 gains over GRPO across all ten settings. NOPO improves over GRPO at both extremes of q_{pos} (19.7–77.5%): non-falsified support scales negative updates rather than directly reinforcing verifier-positive responses. The clearest association is with negative recall: NOPO gains 2.95–6.72 points in the eight code/logic settings, where verifiers detect at least a third of true failures. On OlympiadBench, recall is only 9–13%, leaving few trusted negatives and reducing the NOPO–GRPO margins to $+0.89$ and -0.45 points.

H Comparison with NPO

Table 10 extends the NPO comparison to all five benchmarks on Qwen3-8B under the main experimental protocol. NOPO improves over NPO, the $\gamma = 0$ special case of Eq. 10, on every benchmark by 1.19–4.37 points, supporting the contribution of the non-falsified support weight. NPO also outperforms GRPO on average (49.35 versus 48.27).

For a rollout group with verifier-negative fraction p, the total suppression weight before the common $1/G$ normalization scales as pG for NPO and $p(1-p)G$ for NOPO with $\gamma = 1$. NPO therefore applies the largest suppression to all-failed groups, whereas NOPO's weight is maximal for mixed groups and vanishes at both extremes. For example, groups with 80% and 20% negative rollouts receive equal total weight under NOPO, while NPO assigns four times as much to the former.

Table 9: Pre-RL verifier statistics and the NOPO–GRPO pass@16 gap. Parentheses give the relative improvement over GRPO.

Benchmark	Backbone	q_{pos}	r_{neg}	GRPO	NOPO	NOPO–GRPO
CodeContests	4B	20.7	91.3	43.75	50.47	+6.72 (+15.4%)
CodeContests	8B	19.7	91.4	49.58	55.04	+5.46 (+11.0%)
LCB	4B	24.0	89.2	36.17	39.61	+3.44 (+9.5%)
LCB	8B	25.9	88.0	37.14	42.86	+5.72 (+15.4%)
TACO	4B	64.5	87.9	15.89	19.52	+3.63 (+22.8%)
TACO	8B	66.8	82.5	17.03	19.98	+2.95 (+17.3%)
ZebraLogic	4B	73.0	34.8	74.38	79.37	+4.99 (+6.7%)
ZebraLogic	8B	77.5	56.2	60.62	66.25	+5.63 (+9.3%)
OlympiadBench	4B	46.3	12.5	74.44	75.33	+0.89 (+1.2%)
OlympiadBench	8B	44.4	8.8	76.97	76.52	−0.45 (−0.6%)

Table 10: Direct NPO comparison on Qwen3-8B. GRPO and NOPO results are reproduced from Table 2.

Method	TACO	CodeContests	LCB	ZebraLogic	Olympiad	Avg.
GRPO	17.03	49.58	37.14	60.62	76.97	48.27
NPO	18.16	52.52	38.86	61.88	75.33	49.35
NOPO	19.98	55.04	42.86	66.25	76.52	52.13

I Verifier Implementation Details

This section specifies the two non-trivial verifiers used in the main results: the ZebraLogic constraint-check verifier (Section I.1) and the LLM self-verifier on OlympiadBench (Section I.2).

I.1 Constraint-Check Verifier on ZebraLogic

ZebraLogic puzzles ask the model to assign $K-1$ features to N houses subject to a list of natural-language clues; the gold solution is a JSON object `{"header": [...], "rows": [[...]]}` with exactly N rows. Our verifier checks whether the candidate assignment satisfies the puzzle's *schema and domain constraints* — necessary conditions induced by the task specification — without consulting the clues or the gold assignment.

I.2 LLM Self-Verifier on OlympiadBench

For OlympiadBench, the policy and the verifier use the *same* Qwen3-4B or Qwen3-8B backbone in non-thinking mode.

Three-turn audit prompt. For each rollout we send a three-turn chat:

1. `user`: the original problem statement, with the standard instruction `Please reason step by step, and put your final answer within \boxed{}`.
2. `assistant`: the candidate solution emitted by the policy.
3. `user`: the verification instruction shown below, which asks the verifier to search the existing reasoning for concrete mathematical errors and emit a final-line trailer of the form `Is the existing reasoning correct (Yes/No)? X`.

The instruction explicitly tells the verifier not to demand proof completeness and to output `No` only when it finds a concrete answer-changing mathematical error — a deliberate design choice that makes the verifier behave as a falsifier rather than a certifier.

Prompt Template for the LLM Self-Verifier

```
[User]
{QUESTION}

[Assistant]
{CANDIDATE_REASONING_OR_SOLUTION}

[User]
You are a skeptical but fair math verifier. Audit the student's solution for
concrete mathematical errors, not for proof completeness.

Look for:
- arithmetic mistakes,
- invalid algebraic transformations,
- misuse of a formula or theorem,
- omitted cases or ignored constraints only when you can name a concrete
alternative case that changes the final answer,
- a final answer that contradicts the preceding work or relies on a
demonstrably false step.

Do not mark a solution wrong for style, brevity, missing explanatory prose,
incomplete rigor, or failure to prove optimality, unless you can identify a
concrete counterexample or a specific answer-changing mathematical error.

Verdict policy:
- If you find at least one specific answer-changing error, judge No.
- If the final answer is correct and the computations used are correct, judge
Yes even if the proof is incomplete.
- If a suspected issue is ambiguous, judge Yes rather than hallucinating an
error.

At the end of your response, on a new line, write exactly:
"Is the existing reasoning correct (Yes/No)? X"
where X is either Yes or No.
```

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: The abstract and Section 1 state the OSV setting, the NOPO method, and the empirical scope of the claims. The experimental claims are supported by Sections 3 and 5, with limitations discussed in Section 7.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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: Section 7 discusses the inherent delicacy of negative-only learning and the lack of explicit positive direction. Sections 3 and 5 also identify the low-recall LLM self-verifier regime as a setting where NOPO has a weaker training signal.

   Guidelines:

   - The answer [N/A] 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: [N/A]

Justification: The paper introduces definitions and optimization objectives but does not claim formal theoretical theorems requiring proofs.

Guidelines:

- The answer [N/A] 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: Section 5 describes the experimental setup; Appendices C.1 and C.2 provide the benchmark and training details, and Appendix I describes the verifier implementations. The submitted supplemental artifact will include code, configurations, and instructions for reproducing the experiments.

Guidelines:

- The answer [N/A] 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 submitted supplemental artifact will include code, experiment configurations, and reproduction instructions. The experiments use publicly available benchmarks listed in Appendix C.1.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.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://neurips.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) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix C.1 specifies the benchmark versions and subsets, while Appendix C.2 specifies the optimizer, rollout settings, sequence lengths, method-specific hyperparameters.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [No]

Justification: We do not report multi-seed error bars or formal statistical tests because full RL training runs are computationally expensive, as is common in comparable large-model RL studies. To reduce evaluation variance for the scaling analysis, Table 3 evaluates selected checkpoints with 512 sampled rollouts per problem.

Guidelines:

- The answer [N/A] 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: Appendix C.2 states that each RL run used eight NVIDIA A100 GPUs with 80GB memory and took roughly 12 hours. The same appendix describes the FSDP and vLLM training setup.

   Guidelines:

   - The answer [N/A] 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 authors have reviewed the NeurIPS Code of Ethics and confirm that the research conforms to it.

   Guidelines:

   - The answer [N/A] 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: Section A discusses broader impacts, including the potential benefit of reducing the need for complete verifiers and the risk that stronger reasoning and code-generation models can be misused or optimized toward incomplete checks.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] 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., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper does not involve any release with significant societal impact.

    Guidelines:

    - The answer [N/A] 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: The paper cites the public models, datasets, and software frameworks used in the experiments, and the submitted supplemental artifact will include license and provenance notes for these assets. We use these assets according to their public licenses and terms.

Guidelines:

- The answer [N/A] 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 submitted supplemental artifact will document the code, configurations, and trained checkpoints as new research artifacts.

Guidelines:

- The answer [N/A] 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: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] 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: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects, so IRB approval is not applicable.

Guidelines:

- The answer [N/A] 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 rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: Appendix I.2 describes the LLM self-verifier used as a training-reward component on OlympiadBench and provides its full prompt. LLMs also assisted with manuscript writing.

Guidelines:

- The answer [N/A] 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 in the NeurIPS handbook for what should or should not be described.