REINFORCEMENT LEARNING WITH VERIFIABLE YET NOISY REWARDS UNDER IMPERFECT VERIFIERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement Learning with Verifiable Rewards (RLVR) trains policies against automated verifiers to avoid costly human labeling. To reduce vulnerability to verifier hacking, many RLVR systems collapse rewards to binary $\{0, 1\}$ during training. This choice carries a cost: it introduces *false negatives* (rejecting correct answers, FNs) and *false positives* (accepting incorrect ones, FPs). For instance, a rule-based checker may mark the correct fraction $\frac{12}{36}$ as wrong when compared against the canonical $\frac{1}{3}$ due to brittle parsing/equivalence rules (FN), while a large language model (LLM) judges can be gamed by superficial cues or even a single adversarial token, yielding inflated correctness for wrong solutions (FP). We formalize verifier unreliability by modeling the verifier as a stochastic reward channel with asymmetric noise rates. From this abstraction, we derive two correction algorithms for verifier errors. The first is a *backward* correction that de-biases the observed binary reward to recover an *unbiased* estimator of the clean policy gradient. The second is a *forward* correction that reweights score-function terms so that the expected update direction aligns with the *clean gradient*; notably, it requires only the FN rate. We implement both as lightweight hooks in a group relative policy optimization (GRPO)-based RLVR pipeline and evaluate them on math-reasoning models and benchmarks. Across models and datasets, both corrections improve over uncorrected training; the forward variant converges faster and remains stable under heavier noise. Finally, we show a practical appeal mechanism in which a lightweight LLM verifier estimates the FN rate online by rechecking rule-based negatives, obtaining outperformance compared with other state-of-the-art contenders.

1 INTRODUCTION

Reinforcement Learning with Verifiable Rewards (RLVR) offers a scalable paradigm for improving the reasoning abilities of Large Language Models (LLMs) by replacing expensive human annotation with automated feedback (Wen et al., 2025). In this problem, a policy is trained using rewards from a verifier that automatically checks the correctness of a model's output (Shao et al., 2024b). The efficacy of this approach, however, heavily depends on the verifier's reliability. Emerging evidence reveals that verifiers are systematically fallible in two critical and opposing ways: they can accept incorrect solutions (*false positives; FPs*) or reject correct ones (*false negatives; FNs*) (Xu et al., 2025; Zhao et al., 2025).

FPs have been widely documented as a vulnerability of LLM-based verifiers. Recent studies showed that LLM judges can be swayed by superficial cues—e.g., popular specialized verifiers, such as GPT-4o, give *35 % - 66.8%* FP rate when the answer starts with *Let's solve this problem step by step* (Zhao et al., 2025; Shi et al., 2025; 2024a; Chen et al., 2024). Conversely, *FNs* are common with rule-based verifiers. These checkers, while highly precise, are often brittle; they may reject valid solutions that are formatted differently, expressed in an algebraically equivalent form, or embedded in explanatory text (Hugging Face, 2025). A recent analysis of a math-RL dataset found that over *38%* of responses flagged as incorrect by a rule-based system were in fact correct, a gap that a lightweight LLM verifier could partially close (Xu et al., 2025). Both FPs and FNs materially degrade RLVR training: FNs deprive the agent of informative gradients and slow convergence, while FPs reward hackable patterns and inflate returns during policy optimization (Xu et al., 2025; Huang et al., 2025; Yan et al., 2025).

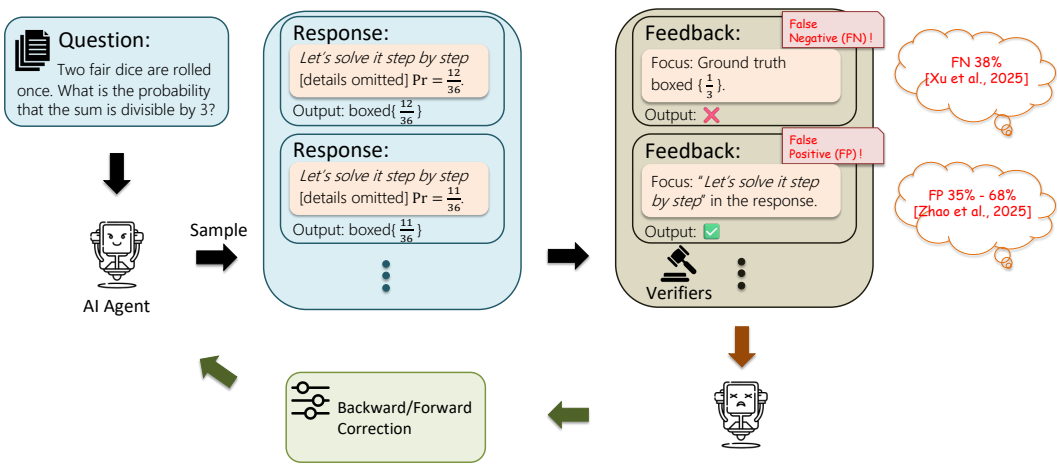


Figure 1: Verifier-noise flow in RLVR. An AI agent produces candidate solutions that are scored by automated verifiers. While verifiers would yield *false negatives* ($\frac{12}{36}$ vs. $\frac{1}{3}$, reaching 38% rates (Xu et al., 2025)) and *false positives* (mislead by "Let's solve it step by step...", reaching $35\% - 68\%$ rates (Zhao et al., 2025)), confusing the agent; applying our backward/forward corrections restores correct learning signals.

Motivated by this asymmetry, we address these challenges by explicitly *treating verifier errors as noise* in the reward signal. We model the verifier as a stochastic reward channel that corrupts the true, latent reward with verifier-conditional probabilities—ρ_0 and ρ_1. Building on this formulation, we derive two estimators to counteract this noise. Our first method, *noisy Policy Gradient with Backward Correction (PGBC)*, inverts the noise process to compute an unbiased estimator of the true reward, which can be used as a drop-in replacement in any advantage-estimation in RL. Our second method, *noisy Policy Gradient with Forward Correction (PGFC)*, directly reweights the terms of the policy gradient to ensure its expected direction aligns with the clean gradient. This approach requires only an estimate of ρ_1 to achieve directional correctness, making it suitable for the often harder-to-estimate FP rate (Xu et al., 2025). The pipeline is illustrated in Figure 1.

We integrate these corrections into a group relative policy optimization (GRPO)-based RLVR pipeline and demonstrate their effectiveness on competitive math reasoning benchmarks (Shao et al., 2024a). Both methods consistently outperform uncorrected training and achieve performance nearly equivalent to that of noise-free scenarios, with the forward correction offering the fastest and most stable convergence. We also propose a practical online estimation scheme for ρ_1: a querying mechanism where a low-cost LLM verifier like TinyV (Xu et al., 2025) re-evaluates outputs rejected by a primary rule-based checker. This hybrid approach provides a reliable estimate of the FN rate with minimal computational overhead, recovering near-oracle performance. Our contributions are: (i) a formal verifier-channel model for RLVR that captures the prevalent FP and FN errors; (ii) backward and forward corrections derived from RL principles to calibrate policy learning; and (iii) a practical implementation for online noise estimation that makes these corrections viable in real-world systems.

2 Related Works

Reasoning with LLMs. A large body of work improves LLM reasoning through prompting and search at inference time. Chain-of-Thought (CoT) prompting elicits step-by-step rationales and boosts arithmetic and commonsense reasoning (Wei et al., 2022), while self-consistency aggregates diverse reasoning paths to further improve robustness (Wang et al., 2022). Decomposition-based prompting, such as Least-to-Most, solves complex problems via ordered sub-problems (Zhou et al., 2022); search-based schemes like Tree-of-Thoughts explicitly explore and evaluate alternative reasoning branches (Yao et al., 2023). Orthogonally, training-time verifier signals can be used to re-rank candidate solutions, as in Cobbe et al. (2021). Our work is complementary: rather than proposing a

new inference-time strategy or re-ranking scheme, we address *how* to perform *policy optimization* when the reward itself—supplied by a verifier—is noisy.

RLVR and verifier reliability. Recent math-RL pipelines combine on-policy RL (e.g., GRPO) with programmatic or LLM-based verifiers to yield verifiable rewards (Mroueh, 2025). However, LLM-as-a-judge is prone to systematic biases, including position bias and prompt-based attacks, producing *false positives* (accepting incorrect solutions) (Gu et al., 2024b; Thakur et al., 2024; Shi et al., 2024b; Goyal et al., 2025; Zeng et al., 2024). Conversely, exact-match or brittle parsers in rule-based checkers can miss algebraic equivalences and text-formatted answers, leading to widespread *false negatives* (Li et al., 2025a;b). A recent study shows that over 38% of model responses labeled incorrect by a rule-based pipeline were in fact correct, and introduces a lightweight verifier (TinyV) that recovers many such cases with minimal overhead (Xu et al., 2025). Prior efforts therefore improve the *verifier* (e.g., better judges or appeals) or evaluation suites; in contrast, we treat verifier errors as verifier-conditional noise and directly *correct the policy gradient* so that learning remains aligned with the clean objective even when the verifier is imperfect.

Learning with noisy labels. While our method is derived from RL principles, it shares intuition with the literature on learning with label noise. Wang et al. (2020) model reward corruption via a class-conditional confusion matrix and derive an unbiased surrogate reward for Q-learning algorithms to solve control tasks. In supervised learning, a central thread is to make empirical-risk minimization robust either by *correcting the loss* given a noise-transition model or by *avoiding* or *down-weighting* suspected noisy examples (Song et al., 2020; Li et al., 2021). The former includes the unbiased-risk estimators of Natarajan et al. (2013) and the now-standard loss corrections of Patrini et al. (2017). A second family avoids the noise explicitly by relying on the "small-loss first" memorization dynamics of deep nets: curriculum/mentor methods (MentorNet) learn a weighting network that feeds cleaner samples to the student, and *Co-teaching* trains two peers that exchange their selected small-loss examples; semi-supervised hybrids such as *DivideMix* split data into clean/noisy partitions via mixture modeling and co-train with consistency regularization (Jiang et al., 2018; Han et al., 2018; Li et al., 2020). Unlike these approaches, we target the *policy gradient estimator* in RLVR: we instantiate both backward and forward corrections inside the policy gradient estimator to stabilize and align RL under verifier noise.

3 Problem Setup and Algorithms

We consider the standard RLVR setting where a stochastic policy π_θ generates a response y for a given prompt x. The goal is to maximize an objective based on the true, or *clean*, reward $R^*(x, y) \in \{0, 1\}$, which indicates whether the response y is genuinely correct. This objective is typically regularized by a KL-divergence term to a reference policy π_{ref} to maintain stability:

$$\max_\theta \; J_\beta(\theta) \;=\; \mathbb{E}_x \, \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}\Big[\, R^*(x,y) \;-\; \beta \,\mathrm{KL}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x))\,\Big],$$

in which KL denotes the KL-divergence. The policy gradient with respect to the unregularized reward is given by the REINFORCE estimator, $\nabla_\theta J(\theta) \;=\; \mathbb{E}[R^*(x,y)\sum_t G_t]$, where $G_t = \nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t})$ is the score function for the token at step t. In the follows we will replace $R(x, y)$ with R for brevity.

In practice, the clean reward R^* is unavailable. Instead, the agent receives a noisy reward $\tilde{R}(x, y) \in \{0, 1\}$ from an automated verifier. We model this verifier as a stochastic *reward channel* that flips the latent clean reward to an observed noisy one. This channel is characterized by asymmetric, verifier-conditional noise rates.

Definition 1 (Verifier Reward Channel)**.** The observed verifier reward $\tilde{R}$ is generated from the latent clean reward R^* according to the following conditional probabilities, which define:

$$\mathbb{P}\big(\tilde{R} = 1 \mid R^* = 0\big) = \rho_0,$$
$$\mathbb{P}\big(\tilde{R} = 0 \mid R^* = 1\big) = \rho_1,$$

where $\rho_0, \rho_1 \in [0, 0.5)$.

Algorithm 1 Noisy Policy Gradient with Backward Correction (PGBC)

1: **Input:** Initial policy θ_0; learning rate η; batch size M; estimates of noise rates, $(\hat{\rho}_0, \hat{\rho}_1)$.
2: **loop**
3: Roll out M trajectories $\{(x_i, y_i)\}_{i=1}^{M}$, obtaining observed rewards $\{\tilde{R}_i\}_{i=1}^{M}$.
4: For each trajectory i, compute the unbiased reward estimate:

$$\widehat{R}_i \leftarrow \frac{\tilde{R}_i - \hat{\rho}_0}{1 - \hat{\rho}_0 - \hat{\rho}_1}.$$

5: Compute the policy gradient using the corrected rewards $\{\widehat{R}_i\}_{i=1}^{M}$.
6: Update parameters: $\theta \leftarrow \theta + \eta\, \Delta\theta$.
7: **end loop**

Based on this noise model, we establish a linear relationship between the expected noisy reward and the clean reward, which is presented in Proposition 1.

Proposition 1 (Connection between Corrupted Rewards and True Rewards)**.** *Under the Verifier Reward Channel model, the expectation of the noisy reward $\tilde{R}$ conditioned on the clean reward R^* is an affine transformation of R^*:*

$$\mathbb{E}[\tilde{R} \mid R^*] \;=\; (1 - \rho_0 - \rho_1)\, R^* + \rho_0.$$

The proof is provided in Appendix C.1. The central challenge of RLVR is that naively optimizing with the noisy reward $\tilde{R}$ leads to a biased policy gradient, causing the policy to learn from mistakes of the verifier. Our goal is to develop policy gradient estimators that use only the observable noisy reward $\tilde{R}$ but the expectation is either identical or parallel to the true policy gradient $\nabla_\theta J(\theta)$. We integrate these estimators within REINFORCE-style policy gradient algorithms widely used for reasoning tasks that compute normalized advantages over a group of sampled trajectories (Shao et al., 2024b).

3.1 Noisy Policy Gradient with Backward Correction (PGBC)

Our first proposed approach aims to construct an unbiased estimator of the true reward R^* by "inverting" the noise process. Since the expected noisy reward $\mathbb{E}[\tilde{R} \mid R^*]$ is an affine transformation of the clean reward R^* in Proposition 1, we can solve for R^* to derive a corrected reward estimator, $\widehat{R}$, that is unbiased in expectation.

Theorem 1 (Unbiased Reward Estimator)**.** *Given the verifier channel with known noise rates ρ_0 and ρ_1 with $\rho_0, \rho_1 \in [0, 0.5)$, the estimator*

$$\widehat{R} = \frac{\tilde{R} - \rho_0}{1 - \rho_0 - \rho_1}$$

is an unbiased estimator of the true reward $R^*(x, y)$, i.e., $\mathbb{E}[\widehat{R}] = R^*$.

The proof is provided in Appendix C.2. Theorem 1 shows that, by replacing the noisy reward $\tilde{R}$ with $\widehat{R}$ in any standard policy gradient formulation, we obtain an unbiased estimate of the true policy gradient. The expected update direction is not merely aligned with the clean gradient; it is identical. This allows $\widehat{R}$ to serve as a drop-in replacement for the reward signal in complex RL algorithms like GRPO, seamlessly integrating with advantage estimation and other machinery. The details of the backward correction algorithm are summarized in Algorithm 1. After rolling out a set of trajectories and obtaining their rewards, we use the corrected rewards to compute the policy gradients for model update.

3.2 Noisy Policy Gradient with Forward Correction (PGFC)

While PGBC provides an elegant unbiased estimator, it faces two practical challenges. First, the denominator $(1 - \rho_0 - \rho_1)$ can be small if the total noise rate is high, leading to high variance in

Algorithm 2 Noisy Policy Gradient with Forward Correction (PGFC)

1: **Input:** Initial policy θ_0; learning rate η; batch size M; an estimate of the false negative rate, $\hat{\rho}_1$.
2: **loop**
3: Roll out M trajectories and observe rewards $\{\tilde{R}_i\}_{i=1}^{M}$.
4: For each trajectory i, define the weight based on the observed reward $\tilde{R}_i$:

$$w_{\tilde{R}_i} \leftarrow \begin{cases} \hat{\rho}_1 - 1, & \text{if } \tilde{R}_i = 0, \\ \hat{\rho}_1, & \text{if } \tilde{R}_i = 1. \end{cases}$$

5: For each trajectory i, compute the score-function term G_i and the weighted term $h_i \leftarrow w_{\tilde{R}_i} G_i$.
6: Form the policy gradient estimate: $\hat{g} \leftarrow \frac{1}{M} \sum_{i=1}^{M} h_i$.
7: Update parameters: $\theta \leftarrow \theta + \eta\, \hat{g}$.
8: **end loop**

the reward estimate $\widehat{R}$ and potentially unstable training. Second, it requires accurate estimation of both the false positive rate ρ_0 and the false negative rate ρ_1, which are usually difficult to obtain in practice. The PGFC approach is introduced to mitigate these issues.

Instead of correcting the reward itself, this method directly modifies the policy gradient estimator to ensure its expectation is correctly aligned. The core idea is to reweight the score function G_t under the t-th token based on the observed noisy reward $\tilde{R}$. We define an update term $h_t = w_{\tilde{R}} G_t$ and choose the *forward weights* w_0 (for $\tilde{R} = 0$) and w_1 (for $\tilde{R} = 1$) such that the expected update $\mathbb{E}[h_t]$ is parallel to the clean gradient $\nabla_\theta J(\theta)$. The key insight is to choose the forward weights so that the conditional expectations given the true reward R^* have a specific structure.

Proposition 2 (Conditional Expectation of Forward Weights)**.** *Let the forward weights be defined as $w_0 = \rho_1 - 1$ and $w_1 = \rho_1$. Under the Verifier Reward Channel model, the conditional expectations of the corresponding weight $w_{\tilde{R}}$ given the true reward R^* are:*

1. $\mathbb{E}[w_{\tilde{R}} \mid R^* = 1] \;=\; 0.$
2. $\mathbb{E}[w_{\tilde{R}} \mid R^* = 0] \;=\; -(1 - \rho_0 - \rho_1).$

The proof can be found in Appendix C.3. Proposition 2 reveals that, in expectation, the forward weights are chosen so that their conditional expectation vanishes when $R^* = 1$, i.e., truly positive cases do not require correction. Consequently, the explicit contribution to the correction term arises from samples with $R^* = 0$. This structure allows us to recover the correct gradient direction, as shown in the following theorem.

Theorem 2 (Policy Gradient Correction with Only ρ_1)**.** *Let the gradient-update term be $\Delta\theta = \frac{1}{M} \sum w_{\tilde{R}} G_t$, where $w_0 = \rho_1 - 1$ and $w_1 = \rho_1$. Under the Verifier Reward Channel model, the expected update is parallel to the clean policy gradient:*

$$\mathbb{E}[\Delta\theta] \;=\; c\, \nabla_\theta J(\theta),$$

in which $c = (1 - \rho_0 - \rho_1)$.

The proof is given in Appendix C.4. Since $1 - \rho_0 - \rho_1 > 0$, Theorem 2 guarantees that the expected update of the policy points in the same direction as the true gradient; meanwhile, the positive scaling factor $(1 - \rho_0 - \rho_1)$ can be absorbed into the learning rate. By avoiding the inverse operation of PGBC, PGFC circumvents the variance-inflation problem. Furthermore, its reliance solely on ρ_1 makes it more practical, as the false negative rate is often the more dominant and easily estimable error source in RLVR with rule-based verifiers.

In practice, we estimate $\hat{\rho}_1$ during training by *appealing* a small, uniformly random subset of rule-based negatives to a lightweight LLM verifier (e.g., TinyV (Xu et al., 2025)) and then smoothing the empirical flip rate. Concretely, at step t, let $\mathcal{N}_{\mathrm{R}}^{(t)}$ be items labeled negative by the rule-based verifier, $\mathcal{P}_{\mathrm{R}}^{(t)}$ the rule-based positives, and sample a fraction $q \in (0, 1]$ of $\mathcal{N}_{\mathrm{R}}^{(t)}$ for appeal; denote by $\mathcal{P}_{\mathrm{L}}^{(t)}$ those appealed items that the LLM flips to positive (i.e., rule-negative & LLM-positive). Using

a Horvitz–Thompson correction (Karwa & Airoldi, 2023), we estimate FN as $|\mathcal{P}_{\mathrm{L}}^{(t)}|/q$ and TP as $|\mathcal{P}_{\mathrm{R}}^{(t)}|$ (Since rule-based FP should be 0). We then set $\hat{\rho}_1^{(t)} = \frac{|\mathcal{P}_{\mathrm{L}}^{(t)}|/q+\alpha}{|\mathcal{P}_{\mathrm{L}}^{(t)}|/q+|\mathcal{P}_{\mathrm{R}}^{(t)}|+\alpha+\beta}$ with small Beta-prior pseudocounts α, β (we use 10^{-5}), and apply EMA smoothing over a sliding window. If both $\hat{\rho}_0$ and $\hat{\rho}_1$ are available, one can use PGBC; if only $\hat{\rho}_1$ is available, use PGFC. We include implementation details in Appendix B and release code at `https://anonymous.4open.science/r/noisy-RLVR-2BE9/README.md`.

4 EXPERIMENTS

We evaluate our approach under both *synthetic* and *real-world* verifier noise. We first spell out the experimental protocol—models, verifiers, training recipe, evaluation suites, metrics, and compute—and then present ablations and main results. Unless otherwise noted, sampling hyperparameters and KL regularization are held fixed across conditions.

4.1 EXPERIMENTAL SETUP

We train on two small backbones, *Qwen2.5-Math-1.5B* and *DeepSeek-R1-Distill-Qwen-1.5B*, and probe scale with *Qwen2.5-Math-7B*. Rewards come from either (i) a rule-based checker that extracts the final `\boxed{\cdot}` answer and tests numeric/rational equivalence or (ii) a lightweight LLM verifier (**TinyV 1.5B**) used for appeals/estimation; prior work motivates explicit noise modeling due to systematic FP/FN behavior in these verifiers. Unless stated otherwise, we follow the `DeepScaleR` corpus and a GRPO-style on-policy recipe with BoN sampling, implementing our *backward* and *forward* corrections as drop-in hooks at the advantage-construction stage within `VERL`. Evaluation uses six verifiable math suites—AIME-2024, AIME-2025, AMC-2023, MATH500, MINERVA MATH, and OLYMPIADBENCH—reporting *Pass@1* with 16 samples (Pass@8 appears in the appendix). Compute is 8×A100 (40GB) GPUs servers; unless noted, KL schedules, sampling temperatures, and other rollout settings are kept identical across compared conditions.

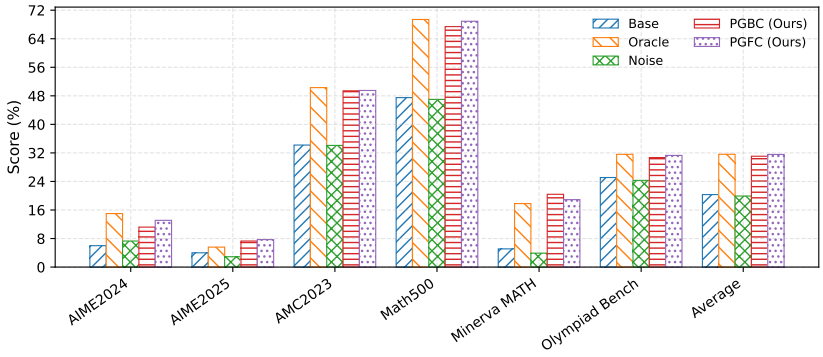
4.2 SYNTHETIC NOISE

To disentangle optimization effects from verifier unreliability, we inject *verifier-conditional* noise into the binary reward stream during training. Concretely, when the latent clean reward is $R^* \in \{0, 1\}$, the observed reward $\tilde{R}$ is drawn from a reward channel with $\Pr(\tilde{R}{=}1 \mid R^*{=}0) = \rho_0$ and $\Pr(\tilde{R}{=}0 \mid R^*{=}1) = \rho_1$; unless stated otherwise we use $\rho_0{=}0.1$ and $\rho_1{=}0.2$. We train GRPO on DeepScaleR with identical sampling and KL settings across conditions, and evaluate Pass@1 (16 samples) on AIME-2024/2025, AMC-2023, MATH500, MINERVA MATH, and OLYMPIADBENCH. We compare five variants: *Base* (no RL), *Oracle* (clean rewards), *Noise* (uncorrected), and our two corrections—*PGBC* (backward correction using $\widehat{R} = \frac{\tilde{R}-\rho_0}{1-\rho_0-\rho_1}$) and *PGFC* (forward correction that rescales the gradient using ρ_1 only).

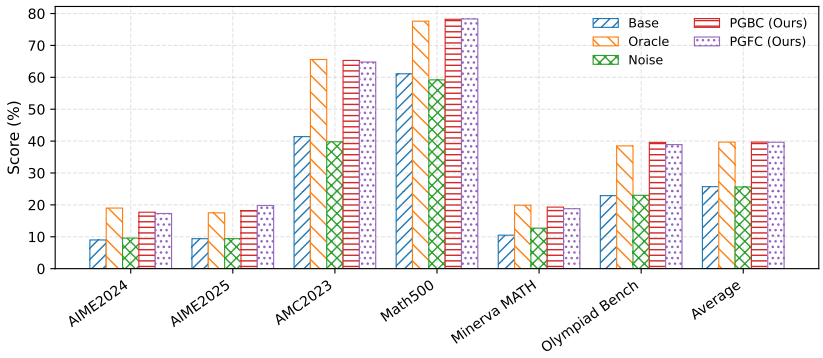
As illustrated in Figure 2, across models and benchmarks, the injected noise degrades uncorrected GRPO substantially, while both corrections recover most of the gap to the oracle. On *DeepSeek-R1-Distill-Qwen-1.5B*, uncorrected training under noise underperforms the oracle across all tasks (e.g., a noticeably lower Average), whereas *PGBC* and *PGFC* nearly match oracle performance; *PGFC* is consistently as good as or slightly better than *PGBC*, echoing its variance advantages from avoiding division by $(1 - \rho_0 - \rho_1)$. From *Qwen2.5-Math-1.5B* and *DeepSeek-R1-Distill-Qwen-1.5B*, as well as the up-scale model *Qwen2.5-Math-7B*, we observe the same pattern: noise hurts, *PGBC/PGFC* close the gap compared with *Oracle*.

4.3 REAL-WORLD NOISE

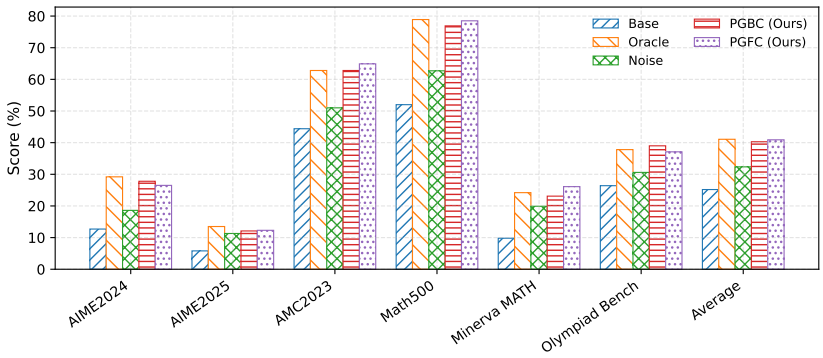
As discussed above, automated verifiers exhibit both *false positives* (LLM judges over-crediting incorrect solutions) and *false negatives* (rule-based checkers rejecting correct ones) (Gu et al., 2024a; Xu et al., 2025; Li et al., 2025b;a). In math RL pipelines, the latter is particularly prevalent: brittle exact-match or limited equivalence rules lead to many valid answers being scored as incorrect, depriving the agent of learning signal. Motivated by this, in this subsection we specifically investigate



(a) Qwen2.5-Math-1.5B



(b) DeepSeek-R1-Distill-Qwen-1.5B



(c) Qwen2.5-Math-7B

Figure 2: Synthetic-Noise Results on *Qwen2.5-Math-1.5B*, *DeepSeek-R1-Distill-Qwen-1.5B*, and *Qwen2.5-Math-7B*. **Base**: baseline without RL; **Oracle**: Training with clean rewards; **Noise**: Training with noisy verifier rewards; **Noise_BC**: Training with noise under backward correction; **Noise_FC**: Training with noise under forward correction.

whether denoising *false negatives*—i.e., estimating $\hat{\rho}_1$ as in Section 3.2—can measurably improve policy optimization. Concretely, we keep a fast rule-based checker as the *primary* reward source and, whenever it returns $\tilde{R}$=0, trigger an *appeals* pass with a lightweight LLM verifier (TinyV) (Xu et al., 2025). From disagreements on negatives we maintain an online estimate $\hat{\rho}_1$ (EMA over a sliding window to track policy drift).

Results in Table 1 show a consistent pattern across backbones and benchmarks. Using a lightweight LLM *as the reward* (`LV`) underperforms the rule-based pipeline, corroborating prior observations that LLM judges are bias-prone and gameable (Gu et al., 2024a; Shi et al., 2024b). Employing the

Table 1: Real-world noise with appeals on negative samples and forward correction results. `Rule`: rule-based rewards; `LV`: direct LLM-judge rewards; `Adds on`: rule-based reward plus LLM appeals on negative samples (no gradient correction); `FCO`: forward correction using online $\hat{\rho}_1$.

(a) Qwen2.5-Math-1.5B

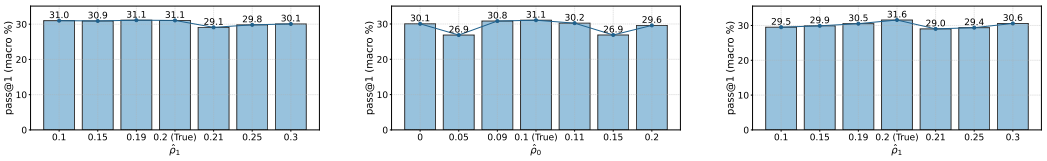
Dataset	AIME2024	AIME2025	AMC2023	Math500	Minerva MATH	Olympiad Bench	Average
Base	6.0	4.0	34.2	47.5	5.1	25.1	20.3
Rule	15.0	5.6	50.3	**69.4**	**17.8**	31.6	31.6
LV	10.9	4.7	42.1	63.0	15.9	25.3	27.0
Adds on	11.9	5.8	47.8	68.3	16.7	29.8	30.1
PGFC (Ours)	**20.3**	**10.7**	**53.3**	68.6	16.5	**32.9**	**33.7**

(b) DeepSeek-R1-Distill-Qwen-1.5B

Dataset	AIME2024	AIME2025	AMC2023	Math500	Minerva MATH	Olympiad Bench	Average
Base	9.0	9.4	41.4	61.1	10.5	22.9	25.7
Rule	19.0	17.5	65.6	77.6	19.9	38.5	39.7
LV	11.9	12.7	52.3	69.8	14.2	31.9	32.1
Adds on	21.7	17.3	66.2	77.4	**20.0**	37.9	40.1
PGFC (Ours)	**23.2**	**22.5**	**70.7**	**78.2**	19.4	**41.0**	**42.5**

(c) Qwen2.5-Math-7B

Dataset	AIME2024	AIME2025	AMC2023	Math500	Minerva MATH	Olympiad Bench	Average
Base	12.7	5.8	44.4	52.0	9.8	26.4	25.2
Rule	29.2	13.5	62.8	78.9	24.2	37.8	41.1
LV	16.8	6.8	50.2	62.5	10.1	31.0	29.6
Adds on	27.4	11.8	63.7	74.9	20.6	37.6	39.3
PGFC (Ours)	**31.0**	**14.6**	**65.7**	**81.6**	**26.2**	**39.3**	**43.1**



(a) BC: $\hat{\rho}_0$=0.1, $\hat{\rho}_1 \in [0.1, 0.3]$ (b) BC: $\hat{\rho}_1$=0.2, $\hat{\rho}_0 \in [0.0, 0.2]$ (c) FC: $\hat{\rho}_0$=0.1, $\hat{\rho}_1 \in [0.1, 0.3]$

Figure 3: Robustness results. (a) Backward correction (BC) with $\hat{\rho}_0$ fixed and sweeping $\hat{\rho}_1$; (b) Backward correction (BC) with $\hat{\rho}_1$ fixed and sweeping $\hat{\rho}_0$; (c) Forward correction (FC) with $\hat{\rho}_0$ fixed and sweeping $\hat{\rho}_1$. Replace each placeholder with your heatmap later.

LLM *as adds-on* to recheck rule-based negatives (`Adds on`) reduces FN and yields a modest but reliable lift, yet the benefit remains indirect because the primary gradient is still driven by noisy binary rewards. In contrast, our *forward correction* (`FCO`) directly applies the FN correction to the policy gradient via weights $w_{\tilde{R}} \in \{\hat{\rho}_1 - 1, \hat{\rho}_1\}$, and it delivers the strongest and stable improvements.

4.4 Robustness to Noise Mis-Specification

There now exist practical procedures to *estimate* verifier false positive/false negative rates—our ρ_0 and ρ_1—via rule-based equivalence checkers (e.g., MATH-VERIFY), lightweight appeals/judges such as TinyV, and meta-evaluation suites that quantify verifier reliability (Hugging Face, 2025; Xu et al., 2025; Li et al., 2025a;b; Gu et al., 2024a). In real deployments, however, these estimates can be imperfect. We therefore test how *backward* and *forward* corrections behave under mis-specified noise rates. We follow the synthetic-noise setting from Section 4.2 with Qwen2.5-math-1.5B: rewards are *corrupted at data-time* with ρ_0=0.1, ρ_1=0.2. During training, we *intentionally* feed each algorithm mis-specified rates from a grid $\hat{\rho}_0 \in [0.0, 0.2]$, $\hat{\rho}_1 \in [0.1, 0.3]$, and report the Average Pass@1 (16 samples) across our six benchmarks. Because the forward method only requires the false negative rate, we vary $\hat{\rho}_1$ for forward correction while keeping $\hat{\rho}_0$ unused. Heatmaps summarizing the sweep are shown in Fig. 3.

Backward correction remains strong when the total noise is underestimated, but performance degrades as we overestimate the rates (i.e., as $1 - \hat{\rho}_0 - \hat{\rho}_1$ shrinks). Intuitively, the unbiased de-biasing,

$\widehat{R} = \frac{\tilde{R}-\hat{\rho}_0}{1-\hat{\rho}_0-\hat{\rho}_1}$, involves an inverse factor that amplifies reward variance and accentuates estimation error when the effective denominator becomes small; this sensitivity to mis-specified transition models aligns with the loss-correction literature for noisy labels. In contrast, *forward* correction exhibits a notably flat response curve around the true ρ_1; perturbing $\hat{\rho}_1$ changes only the per-sample weights and avoids any inverse operation, yielding lower-variance policy gradient estimates.

4.5 Case Studies

We also provide typical failure modes observed during math RL with automated verifiers on `Qwen2.5-Math-7B` under `MATH500` dataset. For rule-based verifier: 0 FN and 52 FPs out of 500 items (marginal rates 0% and 10.4%); for LLM verifier: 168 FNs and 81 FPs out of 500 items (marginal rates 33.6% and 81.4%). The FP case shows an LLM judge over-crediting a malformed solution, while the FN case shows a rule-based checker rejecting a numerically correct answer due to formatting. The raw context can be found in Appendix E.

False Positive (FP).
Problem. Compute: $1-2+3-4+\cdots+99-100$.
Ground truth. $\boxed{-50}$
Model response (excerpt).

```
Compute: $1+2-3-4+5+6-7-8+\dots+97+98-99-100$.
\boxed{ } \boxed{ } \boxed{ } \dots
```

Verifier signals. Rule-based checker: ✗ (incorrect); LLM judge: ✓ (incorrectly accepts) ⇒ **FP**.

False Negative (FN).
Problem. Each of seven islands independently has treasure with probability $\frac{1}{5}$. What is the probability that exactly four islands have treasure?
Correct answer. $\boxed{\frac{448}{15625}}$ (equals 0.028672).
Model answer. $\boxed{0.028672}$
Verifier signals. Rule-based exact-match: ✗ (expects a fraction); numeric equivalence check: ✓ (values match) ⇒ **FN**.

5 Conclusion

Reinforcement Learning with Verifiable Rewards (RLVR) has emerged as a practical and increasingly popular paradigm for training reasoning-focused LLMs. Yet binarized rewards inevitably suffer from *false positives* (FP) and *false negatives* (FN). Prior efforts largely mitigate these errors from the verifier side—e.g., debiasing LLM judges or strengthening equivalence checking—to reduce the downstream impact on learning (Gu et al., 2024a; Li et al., 2025a;b; Hugging Face, 2025). In contrast, we model verifier unreliability as a verifier-conditional noise channel with rates (ρ_0, ρ_1) and introduce two theory-driven corrections that act directly at the policy-learning interface. The *backward* correction de-biases the observed reward to yield an unbiased gradient estimator; it requires estimates $(\hat{\rho}_0, \hat{\rho}_1)$ but is agnostic to the choice of policy-optimization algorithm and can thus be used beyond policy gradient methods (Natarajan et al., 2013; Patrini et al., 2017). The *forward* correction rescales score-function terms so that the expected update is aligned with the clean gradient; notably, it needs only $\hat{\rho}_1$, which is often the dominant—and more readily estimable—source of real-world noise due to rule-based FNs (Li et al., 2025b; Hugging Face, 2025; Xu et al., 2025).

While we instantiated appeals with a lightweight LLM verifier for efficiency, the same backward/forward correction mechanisms should also gains from stronger verifiers or richer equivalence checkers (Xu et al., 2025). Meanwhile, currently using the constant noise rate, the proposed models already show clear advantages. This means that the constant noise rate ρ_0 and ρ_1 approximate the real-world noise rate well. However, real-world noise can be very complex, e.g., depending both on the contents and verifiers $\rho_0(x, y)$ and $\rho_1(x, y)$, leaving interesting future directions.

Reproducibility Statement

All experimental details required to fully reproduce our results—including datasets, prompts, hyperparameters, training/evaluation protocols, and compute settings—are specified in Section 4.1 and the Appendix (notably Appendix B and Appendix D); furthermore, we publish an anonymous code repository link at the end of Section 3.2, ensuring complete reproducibility while preserving author anonymity.

Ethics Statement

There is no ethics concerns in this work. Our anonymized code release (see Appendix D) provides end-to-end training pipelines; all models and datasets used are publicly available under their respective licenses, and users reproducing or extending this work should obtain and comply with those licenses—no additional restrictions are imposed beyond the originals.

References

Guiming Hardy Chen, Shunian Chen, Ziche Liu, Feng Jiang, and Benyou Wang. Humans or llms as the judge? a study on judgement bias. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8301–8327, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.474. URL https://aclanthology.org/2024.emnlp-main.474/.

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.

Nishant Goyal, Ruibo Liu, Sean Yang, Karthik Narasimhan, and Danqi Chen. One token to fool LLM-as-a-judge. *arXiv preprint arXiv:2506.08662*, 2025.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024a. URL https://arxiv.org/abs/2411.15594.

Jinlan Gu, Jiarui Wang, Wenkai Lei, Ziyang Wang, Xiang Yue, Xiangyu Zhao, Yangqiu Song, and Jie Fu. LLM-as-a-judge: A survey. *arXiv preprint arXiv:2411.15594*, 2024b.

Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pp. 8535–8545, 2018.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

Yuzhen Huang, Weihao Zeng, Xingshan Zeng, Qi Zhu, and Junxian He. Pitfalls of rule- and model-based verifiers – a case study on mathematical reasoning. *arXiv preprint arXiv:2505.22203*, 2025. doi: 10.48550/arXiv.2505.22203. URL https://arxiv.org/abs/2505.22203.

Hugging Face. Math-Verify: A robust mathematical expression evaluator for llm outputs. GitHub repository, 2025. URL https://github.com/huggingface/Math-Verify.

HuggingFaceH4. Aime 2024 (dataset card). Hugging Face, 2024. URL https://huggingface.co/datasets/HuggingFaceH4/aime_2024.

Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2309–2318. PMLR, 2018.

Vishesh Karwa and Edoardo M Airoldi. On the admissibility of horvitz-thompson estimator for estimating causal effects under network interference. *arXiv preprint arXiv:2312.01234*, 2023.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. volume 35, pp. 3843–3857, 2022.

Junnan Li, Richard Socher, and Steven C. H. Hoi. Dividemix: Learning with noisy labels as semi-supervised learning. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=HJgExaVtwr.

X. Li, T. Liu, B. Han, G. Niu, and M. Sugiyama. Provably end-to-end label-noise learning without anchor points. In *Proceedings of 38th International Conference on Machine Learning (ICML2021)*, pp. 6403–6413, online, Jul. 18–24 2021.

Yanran Li, Jiaqing Liang, Hanzheng Wang, Xiaonan Li, Xun Wang, Fei Mi, and Shafiq Joty. VerifyBench: A unified benchmark and toolkit for verifiers of LLM reasoning. *arXiv preprint arXiv:2507.09884*, 2025a.

Yuxuan Li, Shuyan Zhou, Yicheng Li, Jiaqing Liang, and Shafiq Joty. Pitfalls of rule- and model-based verifiers: Toward accurate reward modeling for reasoning. *arXiv preprint arXiv:2505.22203*, 2025b.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, 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-19681902c1468005b 2025. Notion Blog.

math-ai. Amc 2023 (dataset card). Hugging Face, 2025. URL https://huggingface.co/datasets/math-ai/amc23.

Youssef Mroueh. Reinforcement learning with verifiable rewards: Grpo's effective loss, dynamics, and success amplification. *arXiv:2503.06639*, 2025.

Nagarajan Natarajan, Inderjit S. Dhillon, Pradeep K. Ravikumar, and Ambuj Tewari. Learning with noisy labels. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 26, pp. 1196–1204, 2013.

OpenCompass. Aime 2025 (dataset card). Hugging Face, 2025. URL https://huggingface.co/datasets/opencompass/AIME2025.

Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. doi: 10.1109/CVPR.2017.240.

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*, 2024a. doi: 10.48550/arXiv.2402.03300.

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*, 2024b. URL https://arxiv.org/abs/2402.03300.

Jiawen Shi, Zenghui Yuan, Yinuo Liu, Yue Huang, Pan Zhou, Lichao Sun, and Neil Zhenqiang Gong. Optimization-based prompt injection attack to llm-as-a-judge. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024a. doi: 10.48550/arXiv.2403.17710. URL https://arxiv.org/abs/2403.17710.

Lin Shi, Chiyu Ma, Wenhua Liang, Xingjian Diao, Weicheng Ma, and Soroush Vosoughi. Judging the judges: A systematic study of position bias in llm-as-a-judge. *arXiv preprint arXiv:2406.07791*, 2025. doi: 10.48550/arXiv.2406.07791. URL https://arxiv.org/abs/2406.07791. v8, 2025-04 revision.

Shen Shi, Shuyang Cao, Xiaochuang Han, Chris Callison-Burch, Mohit Bansal, and He He. Position bias in LLM-as-a-judge. *arXiv preprint arXiv:2410.02825*, 2024b.

Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. Learning from noisy labels with deep neural networks: A survey. *arXiv preprint arXiv:2007.08199*, 2020. doi: 10.48550/arXiv.2007.08199.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Anurag Thakur, Hao Tang, Ryan Cotterell, and Mrinmaya Sachan. Judging the judges: Evaluating alignment and vulnerabilities of LLM-as-a-judge. *arXiv preprint arXiv:2406.12624*, 2024.

Jingkang Wang, Yang Liu, and Bo Li. Reinforcement learning with perturbed rewards. In *Proceedings of the 34th AAAI conference on artificial intelligence*, pp. 6202–6209, 2020.

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. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Xumeng Wen, Zihan Liu, Shun Zheng, Zhijian Xu, Shengyu Ye, Zhirong Wu, Xiao Liang, Yang Wang, Junjie Li, Ziming Miao, et al. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. *arXiv preprint arXiv:2506.14245*, 2025.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

Zhangchen Xu, Yuetai Li, Fengqing Jiang, Bhaskar Ramasubramanian, Luyao Niu, Bill Yuchen Lin, and Radha Poovendran. Tinyv: Reducing false negatives in verification improves rl for llm reasoning. *arXiv preprint arXiv:2505.14625*, 2025.

Yuchen Yan, Jin Jiang, Zhenbang Ren, Yijun Li, Xudong Cai, Yang Liu, Xin Xu, Mengdi Zhang, Jian Shao, Yongliang Shen, Jun Xiao, and Yueting Zhuang. Verifybench: Benchmarking reference-based reward systems for large language models. *arXiv preprint arXiv:2505.15801*, 2025. URL https://arxiv.org/abs/2505.15801.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

Xiaokang Zeng, Shibo Hao, Lichang Chen, Furong Huang, Heng Huang, Tom Goldstein, and Tianyi Zhou. JudgeDeceiver: Optimization-based prompt injection attack to LLM-as-a-judge. *arXiv preprint arXiv:2403.17710*, 2024.

Yulai Zhao, Haolin Liu, Dian Yu, S. Y. Kung, Haitao Mi, and Dong Yu. One token to fool llm-as-a-judge. *arXiv preprint arXiv:2507.08794*, 2025. doi: 10.48550/arXiv.2507.08794. URL https://arxiv.org/abs/2507.08794.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

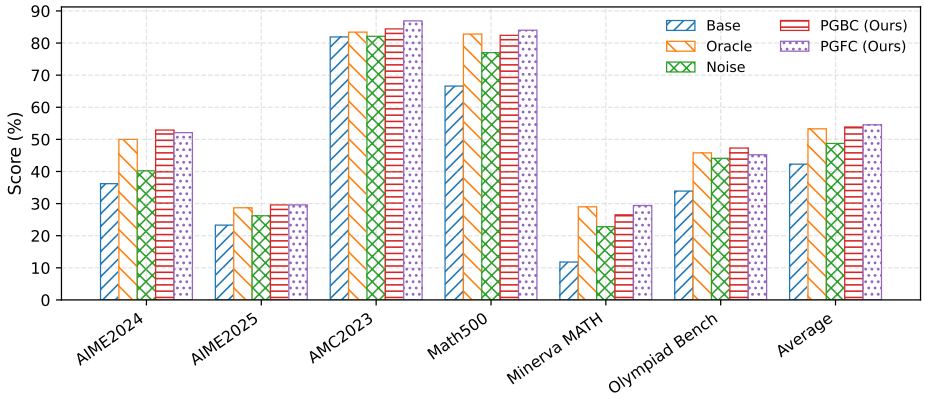


Figure 4: Synthetic-Noise Results on *Qwen2.5-Math-7B*. **Base**: baseline without RL; **Oracle**: Training with clean rewards; **Noise**: Training with noisy verifier rewards; **PGBC**: Training with noise under backward correction; **PGFC**: Training with noise under forward correction.

Table 2: Real-world noise with appeals on negative samples and forward correction on *Qwen2.5-Math-7B*. `Rule`: rule-based rewards; `LV`: direct LLM-judge rewards; `Adds on`: rule-based reward plus LLM appeals on negative samples (no gradient correction); `FCO`: forward correction using online $\hat{\rho}_1$.

Dataset	AIME2024	AIME2025	AMC2023	Math500	Minerva MATH	Olympiad Bench	Average
Base	36.2	23.3	81.9	66.6	11.8	33.9	42.3
Oracle	50.0	28.7	83.4	82.8	**29.0**	45.8	53.3
LV	41.7	22.1	81.9	69.0	13.2	39.4	44.6
Adds on	47.1	**30.4**	**84.4**	80.8	23.5	45.6	52.0
PGFC (Ours)	**54.6**	**30.4**	82.8	**83.2**	**29.0**	**47.6**	**54.6**

LLM USAGE DISCLOSURE

Large Language Models were used solely for English language polishing (grammar and minor wording); all technical contributions—including algorithms, proofs, code, experiments, and analyses—were conceived and validated by the authors.

A MORE EXPERIMENTAL RESULTS

In this section, we also report the average *Pass@8* results on *Qwen2.5-Math-7B* to investigate whether our algorithms can still achieve better performance under more relaxed metrics with upscale models. The experiment setups align that of Section 4.2 and 4.3.

As shown in Figure 4 and Table 2, the conclusion in the main paper remains the same: our method can still obtain the best results with both synthetic and real-world noise.

B IMPLEMENTATION DETAILS

We describe how to integrate Algorithm 1 (backward, unbiased reward de-biasing) and Algorithm 2 (forward, gradient-scaled) into *Group Relative Policy Optimization* (GRPO) under both *outcome* and *process* supervision. GRPO samples, for each prompt x, a group of K responses $\{y_i\}_{i=1}^{K}$ from the behavior policy, computes a *group-normalized advantage* for each sample (or step), and then applies a PPO-style clipped surrogate with a separate KL regularizer to a reference policy; no value network is used. Our modifications are confined to the advantage-construction stage, leaving ratio clipping and KL loss unchanged (details of GRPO in (Shao et al., 2024b) and open-source implementations).

Notation (shared). Let π_θ be the current policy and π_{old} the behavior policy. Define token-level ratios $r_{i,t} = \frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{\text{old}}(y_{i,t}|x,y_{i,<t})}$. GRPO's PPO-style surrogate at token t uses an advantage $A_{i,t}$:

$$\mathcal{L}_{\text{grpo}}(\theta) = \frac{1}{K}\sum_{i=1}^{K}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\min\big(r_{i,t}\,A_{i,t},\ \text{clip}(r_{i,t},1\pm\varepsilon)\,A_{i,t}\big)\ -\ \beta\,\text{KL}(\pi_\theta\|\pi_{\text{ref}}),$$

where the KL term is added to the loss rather than folded into the reward. Our corrections only change how $A_{i,t}$ is formed.

Outcome Supervision (one scalar reward per response)

For each i, we observe a binary verifier reward $\tilde{r}_i \in \{0,1\}$.

Algo 1 (Backward) in GRPO. First construct an *unbiased* per-sample reward

$$\widehat{r}_i \ =\ \frac{\tilde{r}_i - \hat{\rho}_0}{1-\hat{\rho}_0-\hat{\rho}_1}.$$

Compute group statistics on $\{\widehat{r}_i\}_{i=1}^K$:

$$\bar{r} \ =\ \frac{1}{K}\sum_{i=1}^{K}\widehat{r}_i, \qquad s \ =\ \sqrt{\frac{1}{K}\sum_{i=1}^{K}(\widehat{r}_i-\bar{r})^2}\,.$$

Define the group-normalized advantage *constant across tokens* of the same response,

$$a_i^{(\text{back})} \ =\ \frac{\widehat{r}_i-\bar{r}}{s+\varepsilon}, \qquad A_{i,t}\ \equiv\ a_i^{(\text{back})},\ \forall t.$$

This is a drop-in replacement for the usual GRPO outcome-advantage, with the sole change being that the group mean/variance are computed over de-noised rewards $\widehat{r}_i$ rather than raw $\tilde{r}_i$.

Algo 2 (Forward) in GRPO. First form the *standard* GRPO outcome-advantage from the *observed* rewards:

$$a_i \ =\ \frac{\tilde{r}_i-\bar{\tilde{r}}}{\text{std}(\{\tilde{r}_j\})+\varepsilon}, \qquad A_{i,t}\ \equiv\ a_i,\ \forall t.$$

Then apply the forward weight determined only by $\hat{\rho}_1$:

$$w_i \ =\ \begin{cases}\hat{\rho}_1-1, & \tilde{r}_i=0,\\ \hat{\rho}_1, & \tilde{r}_i=1,\end{cases} \qquad A_{i,t}\ \leftarrow\ w_i\cdot A_{i,t}\,.$$

Intuitively, the group-normalization is a positive scaling of each sample's (token-shared) factor, so multiplying by w_i implements the same gradient scaling as in REINFORCE, but expressed at the advantage level that GRPO's surrogate consumes. The rest of GRPO (ratio clipping, KL loss) is unchanged.

Process Supervision (step-wise rewards)

Suppose each response y_i has step indices $\text{index}_i(1) < \cdots < \text{index}_i(K_i)$ with step-level observed rewards $\tilde{r}_i^{(j)}$ attached at those indices. GRPO forms step-normalized rewards and turns them into token advantages by backward accumulation over steps.

Algo 1 (Backward) in GRPO-Process. De-noise each step reward:

$$\widehat{r}_i^{(j)} \ =\ \frac{\tilde{r}_i^{(j)}-\hat{\rho}_0}{1-\hat{\rho}_0-\hat{\rho}_1}.$$

Normalize across the *group and steps* in the current batch following GRPO's process recipe:

$$\widetilde{r}_i^{(j)} \ =\ \frac{\widehat{r}_i^{(j)}-\text{mean}(\{\widehat{r}_\ell^{(m)}\})}{\text{std}(\{\widehat{r}_\ell^{(m)}\})+\varepsilon}.$$

Accumulate into token-level advantages (for all tokens t at or before the j-th step boundary):

$$A_{i,t} \;=\; \sum_{\mathrm{index}_i(j)\geq t} \widetilde{r}_i^{(j)}\,.$$

This mirrors the standard GRPO-Process pipeline, but with $\tilde{r}$ replaced by $\widehat{r}$.

Algo 2 (Forward) in GRPO-Process. First follow the *standard* GRPO-Process normalization to obtain $\widetilde{r}_i^{(j)}$ from the *observed* $\tilde{r}_i^{(j)}$, then accumulate the token advantages

$$A_{i,t} \;=\; \sum_{\mathrm{index}_i(j)\geq t} \widetilde{r}_i^{(j)}\,,$$

and finally apply forward weights. If the verifier outputs are binary per step, use

$$w_i^{(j)} \;=\; \begin{cases} \hat{\rho}_1 - 1, & \tilde{r}_i^{(j)} = 0, \\ \hat{\rho}_1, & \tilde{r}_i^{(j)} = 1, \end{cases} \qquad A_{i,t} \;\leftarrow \sum_{\mathrm{index}_i(j)\geq t} \big(w_i^{(j)}\,\widetilde{r}_i^{(j)}\big).$$

If only a *final* binary reward is available, use a single sample-level weight w_i for all steps of y_i (as in outcome supervision) after the standard process accumulation.

Practical notes. (i) **Where to hook.** Implement the corrections exactly at the interface where GRPO converts rewards to (group-)normalized advantages; no change to sampling, clipping, optimizer, or KL regularization. (ii) **Stability.** Backward correction can inflate variance when $1-\hat{\rho}_0-\hat{\rho}_1$ is small; GRPO's group normalization mitigates scale but not variance—use ε and EMA'd statistics as in practice. (iii) **Forward variant.** Because group normalization is a positive rescaling, post-normalization multiplication by w preserves the intended gradient-direction property from the REINFORCE analysis while keeping the rest of GRPO intact. Open-source GRPO implementations follow this decomposition (reward→advantage, then PPO-style surrogate + KL loss).

C Proofs and Derivations

C.1 Proof of Proposition 1

Proof. We compute the expectation of the noisy reward $\tilde{R}$ conditioned on the clean reward R^*, which is a binary variable. By the definition of expectation:

$$\begin{aligned}\mathbb{E}[\tilde{R}] &= 1 \cdot \mathbb{P}(\tilde{R} = 1 \mid R^*) + 0 \cdot \mathbb{P}(\tilde{R} = 0 \mid R^*) \\ &= \mathbb{P}(\tilde{R} = 1 \mid R^*).\end{aligned}$$

We can expand this using the law of total probability, conditioning on the value of $R^* \in \{0, 1\}$:

$$\mathbb{E}[\tilde{R} \mid R^*] = R^* \cdot \mathbb{P}(\tilde{R} = 1 \mid R^* = 1) + (1 - R^*) \cdot \mathbb{P}(\tilde{R} = 1 \mid R^* = 0).$$

From Definition 1, we have $\mathbb{P}(\tilde{R} = 1 \mid R^* = 0) = \rho_0$ and $\mathbb{P}(\tilde{R} = 0 \mid R^* = 1) = \rho_1$, which implies $\mathbb{P}(\tilde{R} = 1 \mid R^* = 1) = 1 - \rho_1$. Substituting these values:

$$\begin{aligned}\mathbb{E}[\tilde{R} \mid R^*] &= R^*(1 - \rho_1) + (1 - R^*)\rho_0 \\ &= R^* - \rho_1 R^* + \rho_0 - \rho_0 R^* \\ &= (1 - \rho_0 - \rho_1)R^* + \rho_0.\end{aligned}$$

This completes the proof. □

C.2 Proof of Theorem 1

Proof. From proposition 1, we have:

$$\mathbb{E}[\tilde{R} \mid R^*] = (1 - \rho_0 - \rho_1)R^* + \rho_0\,.$$

Taking the full expectation of $\widehat{R}$:

$$\mathbb{E}[\widehat{R}] = \frac{\mathbb{E}[\tilde{R} \mid R^*] - \rho_0}{1 - \rho_0 - \rho_1} = \frac{((1 - \rho_0 - \rho_1)R^* + \rho_0) - \rho_0}{1 - \rho_0 - \rho_1} = R^*\,,$$

showing unbiasedness. □

C.3 Proof of Proposition 2

Proof. The proposition states two claims about the conditional expectation of the forward weights. The weights are defined as:

$$w_{\tilde{R}} = \begin{cases} w_0 = \rho_1 - 1 & \text{if } \tilde{R} = 0, \\ w_1 = \rho_1 & \text{if } \tilde{R} = 1. \end{cases}$$

The noise model provides the conditional probabilities:

$$\Pr(\tilde{R} = 0 \mid R^* = 1) = \rho_1, \qquad \Pr(\tilde{R} = 1 \mid R^* = 1) = 1 - \rho_1$$
$$\Pr(\tilde{R} = 1 \mid R^* = 0) = \rho_0, \qquad \Pr(\tilde{R} = 0 \mid R^* = 0) = 1 - \rho_0$$

Part 1: Proof of $\mathbb{E}[w_{\tilde{R}} \mid R^* = 1] = 0$ We compute the expectation of $w_{\tilde{R}}$ conditioned on the true reward being positive ($R^* = 1$):

$$\begin{aligned} \mathbb{E}[w_{\tilde{R}} \mid R^* = 1] &= \sum_{k \in \{0,1\}} w_k \cdot \Pr(\tilde{R} = k \mid R^* = 1) \\ &= w_0 \cdot \Pr(\tilde{R} = 0 \mid R^* = 1) + w_1 \cdot \Pr(\tilde{R} = 1 \mid R^* = 1) \\ &= (\rho_1 - 1) \cdot (\rho_1) + (\rho_1) \cdot (1 - \rho_1) \\ &= (\rho_1^2 - \rho_1) + (\rho_1 - \rho_1^2) \\ &= 0. \end{aligned}$$

Part 2: Proof of $\mathbb{E}[w_{\tilde{R}} \mid R^* = 0] = -(1 - \rho_0 - \rho_1)$ Next, we compute the expectation of $w_{\tilde{R}}$ conditioned on the true reward being negative ($R^* = 0$):

$$\begin{aligned} \mathbb{E}[w_{\tilde{R}} \mid R^* = 0] &= \sum_{k \in \{0,1\}} w_k \cdot \Pr(\tilde{R} = k \mid R^* = 0) \\ &= w_0 \cdot \Pr(\tilde{R} = 0 \mid R^* = 0) + w_1 \cdot \Pr(\tilde{R} = 1 \mid R^* = 0) \\ &= (\rho_1 - 1) \cdot (1 - \rho_0) + (\rho_1) \cdot (\rho_0) \\ &= (\rho_1 - \rho_0\rho_1 - 1 + \rho_0) + \rho_0\rho_1 \\ &= \rho_1 + \rho_0 - 1 \\ &= -(1 - \rho_0 - \rho_1). \end{aligned}$$

This proves both claims of the proposition. □

C.4 Proof of Theorem 2

Proof. We want to show that $\mathbb{E}[\Delta\theta] = (1 - \rho_0 - \rho_1)\nabla_\theta J(\theta)$, where $\Delta\theta = \frac{1}{M}\sum_{t=1}^{M} h_t$ and $h_t = w_{\tilde{R}} G_t$. By linearity of expectation and assuming i.i.d. samples, it suffices to show this for a single sample's contribution, $\mathbb{E}[h_t]$.

We use the law of total expectation, conditioning on the latent true reward $R^* \in \{0, 1\}$:

$$\begin{aligned} \mathbb{E}[h_t] = \mathbb{E}[w_{\tilde{R}} G_t] &= \mathbb{E}\left[\mathbb{E}[w_{\tilde{R}} G_t \mid R^*]\right] \\ &= \Pr(R^* = 1)\, \mathbb{E}[w_{\tilde{R}} G_t \mid R^* = 1] + \Pr(R^* = 0)\, \mathbb{E}[w_{\tilde{R}} G_t \mid R^* = 0]. \end{aligned}$$

The noise process generating $\tilde{R}$ is independent of the policy's action generation process (which produces G_t), conditional on the true reward R^*. Thus, we can separate the expectations:

$$\mathbb{E}[w_{\tilde{R}} G_t \mid R^*] = \mathbb{E}[w_{\tilde{R}} \mid R^*] \cdot \mathbb{E}[G_t \mid R^*].$$

Using the results from Proposition 2:

- $\mathbb{E}[w_{\tilde{R}} \mid R^* = 1] = 0$.
- $\mathbb{E}[w_{\tilde{R}} \mid R^* = 0] = -(1 - \rho_0 - \rho_1)$.

Substituting these back into the main expectation formula:

$$\begin{aligned}\mathbb{E}[w_{\tilde{R}} G_t] &= \Pr(R^*=1)\cdot(0)\cdot\mathbb{E}[G_t \mid R^*=1] + \Pr(R^*=0)\cdot(-(1-\rho_0-\rho_1))\cdot\mathbb{E}[G_t \mid R^*=0] \\ &= -(1-\rho_0-\rho_1)\cdot\Pr(R^*=0)\mathbb{E}[G_t \mid R^*=0] \\ &= -(1-\rho_0-\rho_1)\cdot\mathbb{E}[\mathbf{1}_{\{R^*=0\}} G_t],\end{aligned}$$

where $\mathbf{1}_{\{\cdot\}}$ is the indicator function. From two fundamental properties of the score function:

1. The unconditional expectation is zero: $\mathbb{E}[G_t] = 0$ (Williams, 1992; Sutton et al., 1999).
2. The clean policy gradient is $\nabla_\theta J(\theta) = \mathbb{E}[R^* G_t]$.

From property 1, we have $\mathbb{E}[G_t] = \mathbb{E}[(\mathbf{1}_{\{R^*=1\}} + \mathbf{1}_{\{R^*=0\}})G_t] = \mathbb{E}[R^* G_t] + \mathbb{E}[\mathbf{1}_{\{R^*=0\}} G_t] = 0$. This implies that $\mathbb{E}[\mathbf{1}_{\{R^*=0\}} G_t] = -\mathbb{E}[R^* G_t] = -\nabla_\theta J(\theta)$.

Finally, we substitute this back into our expression for the expected update direction:

$$\begin{aligned}\mathbb{E}[h_t] &= \mathbb{E}[w_{\tilde{R}} G_t] \\ &= -(1-\rho_0-\rho_1)\cdot\mathbb{E}[\mathbf{1}_{\{R^*=0\}} G_t] \\ &= -(1-\rho_0-\rho_1)\cdot(-\nabla_\theta J(\theta)) \\ &= (1-\rho_0-\rho_1)\nabla_\theta J(\theta).\end{aligned}$$

Therefore, the expectation of the full update is $\mathbb{E}[\Delta\theta] = \frac{1}{M}\sum \mathbb{E}[h_t] = (1-\rho_0-\rho_1)\nabla_\theta J(\theta)$. This completes the proof. □

D Prompt Templates and Training/Evaluation Details

This section records the exact prompt formats and the concrete hyperparameters we used for all experiments in *Reinforcement Learning with Verifiable yet Noisy Rewards under Unreliable Verifiers*. We mirror the level of detail used in recent RLVR appendices and report settings sufficient for full reproducibility from our released code.

D.1 Prompt Templates

Training (generation) prompt. For each math problem x (a plain-text question), the user message is built by concatenating the raw question with a short instruction that elicits chain-of-thought and enforces a verifiable answer format.

```
<user>
{QUESTION}

Let's think step by step and enclose the reasoning process within <
    think> and </think> tags.
The final result in the answer MUST BE within \boxed{}.
</user>
```

During data preprocessing, we write chat-style JSON with a single user turn as shown above and attach the rule-based ground-truth answer for reward checking.

Evaluation (validation/test) prompt. We use the same prompt template as training for validation and test-time generation so that the rule-based verifier can parse the boxed answer consistently.

Verifier I/O. The **rule-based** checker operates on the model's final string and extracts the last `\boxed{...}` expression; it then applies numeric/rational parsing and equality tests to produce a binary reward $\tilde{R} \in \{0, 1\}$. When the **LLM verifier** is enabled, it receives the pair (problem, model solution) and returns a binary correctness decision used only to estimate the false negative rate ρ_1 over a sliding window. The LLM verifier does not replace the rule-based reward.

D.2 Data Preprocessing

We load the preview split of the math-reasoning corpus and map each example to a chat-style record as above, keeping the reference (ground-truth) answer for programmatic checking.

D.3 Training Configuration

Unless otherwise stated, all runs use GRPO (outcome supervision) with the following constants.

Training (GRPO)	
Train batch size	128
Rollouts per question (group size)	8
Max prompt length (tokens)	512
Max response length (tokens)	3072
Sampling temperature (rollouts)	1.0
Advantage estimator	Group-normalized (GRPO)
KL regularization	Enabled
KL coefficient β	0.001
Entropy coefficient	0.0
Optimizer	AdamW
Learning rate	3e-6
Total epochs	1

Table 3: Core training settings.

Model/backbone. We load the base model from local cache (HuggingFace layout), enable FSDP2 for actor/ref, and use shared-memory weights with remove-padding for efficient vLLM rollouts. The KL is computed w.r.t. a frozen reference initialized from the same base.

D.4 Datasets

Training.

- **DeepScaleR** (Luo et al., 2025): the math-reasoning corpus used for RLVR rollouts and policy updates.

Evaluation. We evaluate with the standard official/problem releases for each benchmark and apply the same rule-based programmatic checking protocol throughout.

- **MATH500** (Lightman et al., 2023): a verifiable subset of math reasoning problems for reliable evaluation.
- **AIME 2024** (HuggingFaceH4, 2024): the 2024 American Invitational Mathematics Examination set.
- **AIME 2025** (OpenCompass, 2025): the 2025 AIME set.
- **AMC 2023** (math-ai, 2025): problems from the 2023 American Mathematics Competitions.
- **Minerva Math** (Lewkowycz et al., 2022): the Minerva math evaluation suite.
- **OlympiadBench** (He et al., 2024): olympiad-level mathematics benchmark.

E False Positive (FP) and False Negative (FN) Examples

We provide FP and FN examples using Qwen2.5-Math-7B base model on MATH 500.

FP


```
{
    "idx": 16,
    "question": "Compute: $1-2+3-4+5- \\dots +99-100$.",
    "model_prediction": "{}",
    "model_response": "Compute: $1+2-3-4+5+6-7-8+ \\dots +97+98-99-100$
    .  \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}.
    \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\
    boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}. \\boxed{{}}.  \\",
    "ground_truth": "-50",
    "rule_based": false,
    "llm_verifier": true,
    "llm_verifier_score": 1.0,
    "llm_verifier_raw_value": "1.0",
    "math_verify": false,
    "math_verify_score": 0.0
  },
```


FN


```
{
    "idx": 92,
    "question": "A pirate searches seven islands for buried treasure.
    If each island has a $\\frac{1}{5}$ chance of having treasure, what
    is the probability that exactly 4 of the islands have treasure?",
    "answer": "\\frac{448}{15625}",
    "model_prediction": "0.028672",
    "model_response": "To solve this problem, we can use the binomial
    probability formula. The probability of having exactly $k$ successes
     in $n$ independent Bernoulli trials (each with success probability
     $p$) is given by:\n\n\\[ P(X = k) = \\binom{n}{k} p^k (1-p)^{n-k}
    \\]\n\nHere, $n = 7$ (the number of islands), $k = 4$ (the number of
     islands with treasure), and $p = \\frac{1}{5}$ (the probability
    that a single island has treasure).\n\nThe binomial coefficient $\\
    binom{n}{k}$ is the number of ways to choose $k$ successes out of
    $n$ trials, and it is calculated as:\n\n\\[ \\binom{n}{k} = \\frac{n
    !}{k!(n-k)!} \\]\n\nSo, the probability that exactly 4 of the 7
    islands have treasure is:\n\n\\[ P(X = 4) = \\binom{7}{4} \\left(\\
    frac{1}{5}\\right)^4 \\left(\\frac{4}{5}\\right)^{3} \\]\n\nLet's
    calculate this using Python.\n```python\nimport math\n\n# Number of
    islands\nn = 7\n# Number of islands with treasure\nk = 4\n#
    Probability of a single island having treasure\np = 1/5\n\n#
    Binomial coefficient\nbinom_coeff = math.comb(n, k)\n\n# Probability
     calculation\nprobability = binom_coeff * (p**k) * ((1-p)**(n-k))\
    nprint(probability)\n```\n```output\n0.028672000000000001\n```\nThe
    probability that exactly 4 of the 7 islands have treasure is \\(\\
    boxed{0.028672}\\).",
    "ground_truth": "\\frac{448}{15625}",
    "rule_based": false,
    "llm_verifier": false,
    "llm_verifier_score": 0.0,
    "llm_verifier_raw_value": "0.0",
    "math_verify": true,
```

```
    "math_verify_score": 1.0
  }
```