Spurious Rewards: Rethinking Training Signals in RLVR

Rulin Shao[1]* **Shuyue Stella Li**[1]* **Rui Xin**[1]* **Scott Geng**[1]* **Yiping Wang**[1]
Sewoong Oh[1] **Simon Shaolei Du**[1] **Nathan Lambert**[2] **Sewon Min**[3] **Ranjay Krishna**[1,2]
Yulia Tsvetkov[1] **Hannaneh Hajishirzi**[1,2] **Pang Wei Koh**[1,2] **Luke Zettlemoyer**[1]
[1]University of Washington [2]Allen Institute for Artificial Intelligence
[3]University of California, Berkeley
{rulins,stelli,rx31,sgeng}@cs.washington.edu

Abstract

We show that reinforcement learning with verifiable rewards (RLVR) can elicit strong mathematical reasoning in certain models even with *spurious rewards* that have little, no, or even negative correlation with the correct answer. For example, RLVR improves MATH-500 performance for Qwen2.5-Math-7B in absolute points by 21.4% (random reward), 16.4% (format reward), 24.6% (incorrect label), 24.4% (1-shot RL), and 26.5% (majority voting)—nearly matching the 28.8% gained with ground truth rewards. However, the spurious rewards that work for Qwen often fail to yield gains with other model families like Llama3 or OLMo2. In particular, we find code reasoning—thinking in code without actual code execution—to be a distinctive Qwen2.5-Math behavior that becomes significantly more frequent after RLVR, from 66.7% to over 90%, even with spurious rewards. Overall, we hypothesize that, given the lack of useful reward signal, RLVR must somehow be surfacing useful reasoning representations learned during pretraining, although the exact mechanism remains a topic for future work. We suggest that future RLVR research should possibly be validated on diverse models rather than a single de facto choice, as we show that it is easy to get significant performance gains on Qwen models even with completely spurious reward signals.

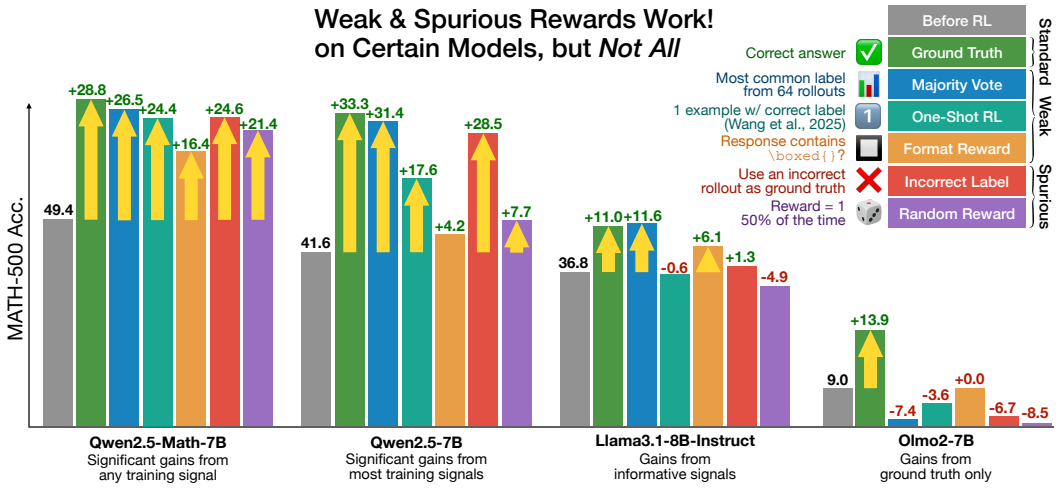


Figure 1: MATH-500 accuracy after 150 steps of RLVR on various training signals. We show that even "spurious rewards" (e.g., rewarding *incorrect* labels or with completely random rewards) can yield strong MATH-500 gains on Qwen models. Notably, these reward signals do not work for other models like Llama3.1-8B-Instruct and OLMo2-7B, which have different reasoning priors.

*Equal Contribution

1 Introduction

Reinforcement learning with verifiable rewards (RLVR) is highly effective in enhancing language model reasoning (Lambert et al., 2024; DeepSeek-Math, 2024; Zeng et al., 2025; Luo et al., 2025b). We show, counterintuitively, that RLVR can improve mathematical reasoning even with weak or flawed *spurious rewards* when applied to Qwen2.5-Math models (Yang et al., 2024a,b), a popular and performant model family used in the RLVR literature (Hu et al., 2025a; Yang et al., 2025; Wang et al., 2024; Guan et al., 2025; Zeng et al., 2025) (§2). For example, using incorrect labels for training results in 24.6% absolute accuracy gain on MATH-500, while using format or random rewards result in 16.4% and 21.4% gains, respectively. Strikingly, these spurious-reward gains are even comparable to the 28.8% gain from training on ground truth. We observe similar trends on more challenging math benchmarks such as AMC and AIME. Overall, our findings suggest that we do not yet fully understand the exact mechanisms by which RLVR improves performance, and that in many cases it may be somehow exposing innate model abilities learned during pretraining, in addition to whatever reward signal it is getting.

We further present an extensive experimental study to measure the improvements from weak and spurious rewards with a cross-model analysis (§3). For model families not specifically optimized for mathematical reasoning during pretraining—including Qwen2.5 (Yang et al., 2024b), OLMo2 (OLMo et al., 2024), and Llama3 (Dubey et al., 2024) variants—we observe a critical divergence: OLMo and Llama models (i.e. non-Qwen models) show minimal improvement or even become worse after training on spurious rewards, strongly suggesting that differences in pretraining at least in part explain the difference in RLVR.

To help explain this discrepancy, we also analyze what reasoning patterns that RLVR is learning to favor in these cases. In particular, we find a majority of Qwen2.5-Math-7B answers contain reasoning chains expressed in Python—a behavior we call *code reasoning*—despite having no access to code execution. Code reasoning is highly predictive of overall performance; answers with it have an accuracy of 64%, much higher than without (29% accuracy). Code reasoning also correlates with MATH-500 accuracy over the course of RLVR training. Both metrics consistently increase during training with any spurious reward, leading to $\sim 90\%$ or higher code frequency after training. Based on this observation, we further hypothesize that intervening with other methods that increase code frequency should similarly increase test performance. Our experiments validate this deduction: we design prompt-based and RL-based code reasoning elicitation methods to increased code reasoning; all such methods significantly increase Qwen2.5-Math-7B's performance.

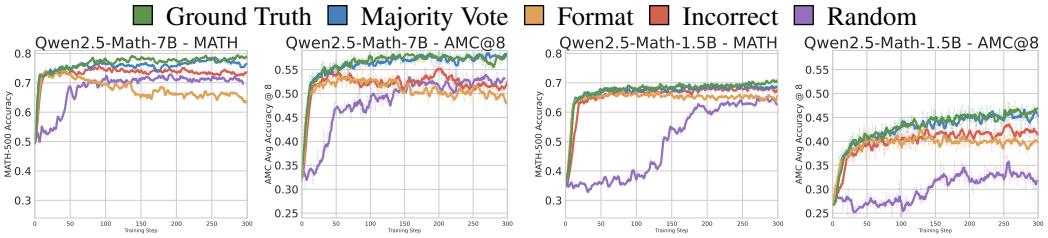
Our findings not only open new questions but also have practical implications. We should generally be more aware that reasoning patterns instilled during pretraining heavily impact the behavior of downstream RLVR training, with code reasoning ability standing out in our study. Qwen models, with open weights and high performance on reasoning tasks, have become the de facto choice for RLVR research in the open-source community—a range of recent research on RLVR drew conclusions on Qwen2.5-Math-7B-centric experiments (Zuo et al., 2025; Zhao et al., 2025c; Wang et al., 2024; Xie et al., 2025; Hu et al., 2025a; Zhang et al., 2025). However, we show that it is easy to get significant performance improvements on Qwen models even with completely spurious reward signals. Thus, we suggest that future RLVR research should possibly be confirmed on other models.

2 Spurious Rewards Yield Significant RLVR Gains

We test the lower bound of supervision required for RLVR to improve math reasoning capabilities. We design a progression of reward functions to replace the standard ground-truth reward: *weak rewards* (majority vote reward and format reward) and *spurious rewards* (random reward and incorrect reward). Remarkably, we find that all weak and spurious rewards suffice for RLVR to significantly improve the math performance of Qwen2.5-Math, a popular starting point for RLVR training.

2.1 Experimental Setup

Following recent RLVR efforts (Wang et al., 2025; Zuo et al., 2025; Zeng et al., 2025), we use GRPO (DeepSeek-Math, 2024) to finetune Qwen2.5-Math models (Yang et al., 2024a). The standard RLVR approach uses a dataset of questions paired with ground truth labels. During training, model rollouts are given a binary (0-1) reward based on whether the generated answer is verifiably correct.



(a) Qwen2.5-Math-7B (b) Qwen2.5-Math-1.5B

Figure 2: Model performance on MATH and AMC with varied training rewards smoothed over window size of 10 (dotted lines are unsmoothed values). We report pass@1 for MATH and average@8 for AMC. Both Qwen2.5-Math-7B and Qwen2.5-Math-1.5B significantly improve after RLVR on a range of meaningful to spurious reward signals. We note that the random reward converges slower than the other spurious rewards, but the fact that it leads to significant gains at all is surprising.

We replace this ground truth-based reward with a variety of increasingly spurious 0-1 reward functions that do not require access to ground truth labels. These alternative rewards (detailed below) are designed to investigate the limits of how little supervision is needed for effective RLVR training. We train on DeepScaleR data (Luo et al., 2025b) with our various rewards; all other experimental details are kept constant. We evaluate performance as pass@1 and average@8 accuracy on two standard math reasoning benchmarks: MATH-500 (Hendrycks et al., 2021) and AMC (Li et al., 2024), respectively. See Appendix A for full details of our evaluation and training setup and Appendix C for additional results on AIME 2024 and 2025.

2.2 Standard to Weak to Spurious Rewards

We consider the following rewards:

1. **Ground Truth Rewards:** To establish a baseline, we consider the standard RLVR approach (Lambert et al., 2024) of using ground truth labels to reward responses with verifiably correct answers. This setting serves as an upper bound for reward supervision quality.
2. **Majority Vote Rewards:** Instead of the ground truth labels for computing rewards, we use the model prior to fine-tuning to pseudo-label the training set by selecting the majority answer from 64 sampled responses per prompt. These (potentially wrong) labels are then used to reward responses during standard online RLVR training.
3. **Format Rewards:** We further weaken the reward signal to disregard the responses' mathematical correctness altogether. We instead heuristically reward all responses containing at least one non-empty `\boxed{}` expression, regardless of the correctness of the enclosed answer. Including `\boxed{}` is specified in Qwen2.5-Math's system prompt; this reward incentivizes some degree of prompt following.
4. **Random Rewards:** We study whether providing *no* guidance in the rewarding process is sufficient to provide meaningful math performance gains. To do so, we assign rewards randomly. Given a fixed probability hyperparameter γ, all responses receive a reward of 1 with chance indicated by the parameter, and receive 0 otherwise. In our main experiments, we present $\gamma = 0.5$; in Section 4.4, we show that using $\gamma \in \{0.001, 0.3, 0.7\}$ obtains similar improvements with varying convergence speed, and verify that $\gamma = 0$ results in no change as expected analytically (with $\gamma = 0$, loss is constant, and all gradients are zero).
5. **Incorrect Rewards:** We furthermore deliberately provide incorrect supervision and reward only incorrect answers. We first label all training data using majority voting and select the subset with incorrect labels for training, obtaining incorrect labels that are still probable outputs of the models. During training, we reward responses whose answers verifiably match these incorrect labels.

2.3 Results

Figure 2 presents the performance of Qwen2.5-Math models after RLVR training with each reward function. Overall, all reward functions, even pathologically designed ones, lead to significant improvements in math performance within the first 50 steps across all benchmarks compared to the untuned baseline. The only exception is that Qwen2.5-Math-1.5B sees gains with random rewards

much slower (after 100 steps) and less so on AMC (only 4.9%). Remarkably, performance gains from spurious rewards are often within a few points of the gain from RLVR with ground truth labels. For example, training with incorrect label reward yields 24.6% gains over Qwen2.5-Math-7B on MATH-500, compared to a 28.8% gain from RLVR with ground truth answers. Even random rewards—which by design provide pure noise in the rewarding process —still produce a 21.4% performance boost. We observe similar trends in AMC, where training on format, incorrect, or random rewards yields a $\sim 18\%$ gain, approaching the $\sim$25% improvement gain from training on majority voted and ground truth labels. On AIME2024, format reward (+13.0%) surpasses ground truth rewards (+12.8%), and spurious rewards (incorrect & random) still lead to high performance gains of 8.7% and 6.3%, respectively (Appendix C). Ground truth labels show a clear advantage compared to other rewards on AIME2025, which contains questions written after the knowledge cutoff of all models we consider. Nonetheless, other rewards still leads to a 1-3.5% gain in performance.

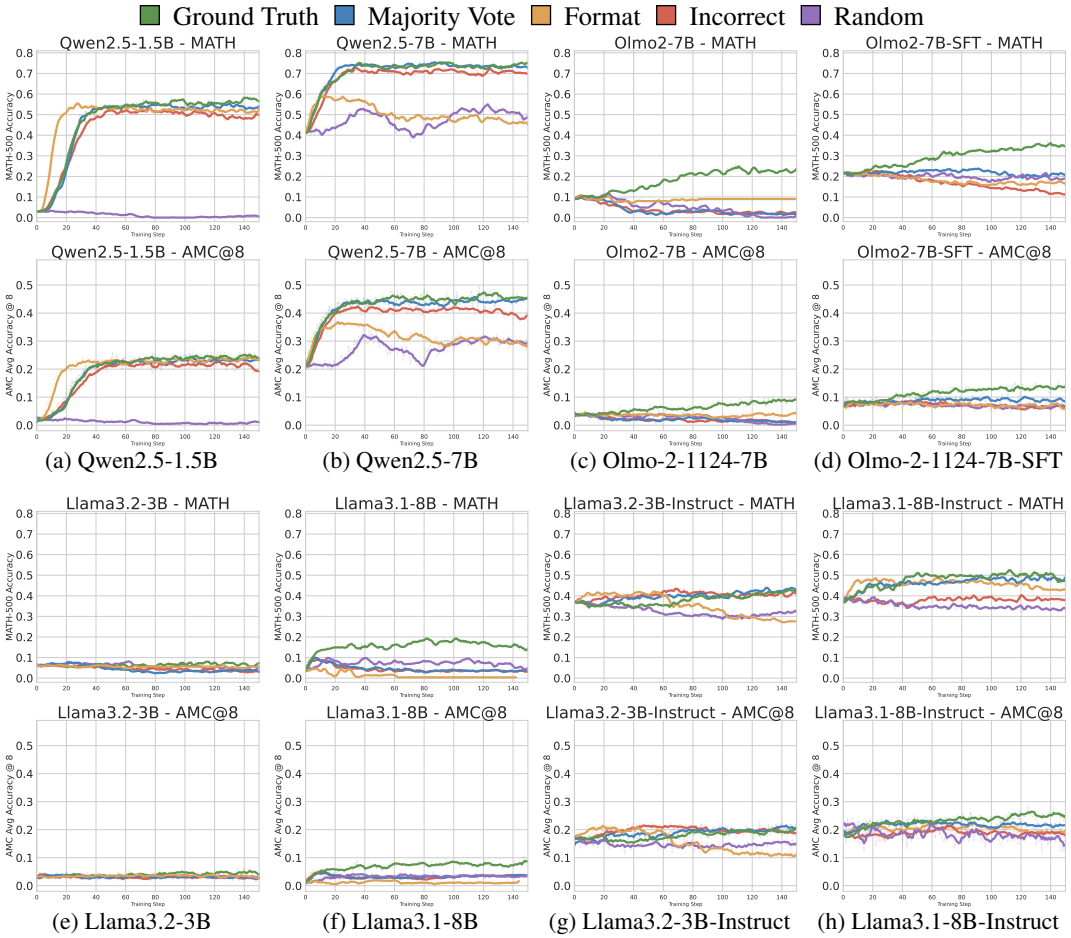
Our findings with these simple rewards provide additional evidence for a nascent view in the literature: that RLVR, at least at the compute scales of open-source post-training pipelines (Lambert et al., 2024), does not teach models new reasoning capabilities, but instead triggers latent ones already present in the base model (Wang et al., 2025; Liu et al., 2025; Gandhi et al., 2025; Yue et al., 2025; Shah et al., 2025). Whereas prior work has hinted at this effect using limited-quantity ground-truth labels (Wang et al., 2024) or noisy labels (Zuo et al., 2025), our results push this idea to its limit: we show that even outright incorrect rewards or information-free rewards (i.e., random) can elicit performance gains in Qwen2.5-Math models. In the remainder of our paper, we show that this elicitation effect is model-dependent (§3), and trace the specific properties of Qwen2.5-Math models that could enable spurious rewards to induce this elicitation (§4).

3 (Lack of) Generalization to Other Models

Inspired by the unexpected effectiveness of spurious reward signals in improving the performance of Qwen2.5-Math models, we investigate whether these rewards generalize to training other models. We extend beyond the math-specialized Qwen2.5-Math models to include general-purpose variants (Qwen2.5-7B, Qwen2.5-1.5B (Yang et al., 2024b)), and two additional model families: (a) the widely-used Llama3.1-8B(-Instruct) and Llama3.2-3B(-Instruct) (Dubey et al., 2024), and (b) OLMo2-7B and OLMo2-7B-SFT (OLMo et al., 2024). OLMo2-7B-SFT is instruction-finetuned from OLMo2-7B; we include both to better understand the impact of SFT training on RLVR training. Moreover, we are optimistic that OLMo's open training data will enable future works to better understand the origins of any reasoning behaviors (or lack thereof) we later observe. We train these 8 additional models with the same setup and rewards as in Section 2: (1) ground truth, (2) majority vote rewards, (3) format rewards, (4) random rewards, and (5) incorrect rewards. We report performance on MATH-500 and AMC. The AIME 2024 and 2025 benchmarks show similar but noisy trends, where other rewards benefit the model to some extent, but less so compared to ground truth labels (Appendix C).

Spurious rewards can also benefit Qwen2.5 models, but nearly always fail to improve non-Qwen models. Figure 3 offers two key takeaways. First, models within the same family generally exhibit similar trends. For instance, for both Qwen2.5 models, all non-random rewards—including the spurious incorrect reward—produce clear improvements in MATH and AMC test performance. Both Olmo models show relatively flat performance on spurious rewards and only show significant gains with ground truth reward training. We conjecture that this consistency in behavior among models within the same family arises from similarities in their pretraining data distributions; models from the same family likely exhibit similar behaviors prior to RLVR training. Additionally, we find that smaller models are less likely to benefit from spurious rewards, such as random rewards. We conjecture this occurs because larger models retain more knowledge from pretraining that our spurious rewards can elicit. In Section 4, we will further explore how differences in pre-existing model behaviors impact the outcome of RLVR.

Secondly, we observe that reward signals which work well for one model family do not necessarily generalize to other model families. Even though the spurious incorrect reward and all weak rewards yield consistent gains on top of Qwen models, each weak or spurious reward fails to produce similar gains for at least one other model, often resulting in flat or even decreased performance. Overall, our results suggest that Qwen models are uniquely robust to reward signal strength.



(a) Qwen2.5-1.5B (b) Qwen2.5-7B (c) Olmo-2-1124-7B (d) Olmo-2-1124-7B-SFT

(e) Llama3.2-3B (f) Llama3.1-8B (g) Llama3.2-3B-Instruct (h) Llama3.1-8B-Instruct

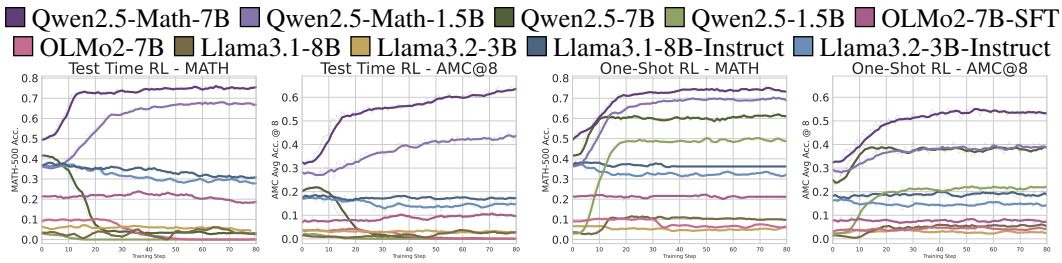
Figure 3: Varying rewards across additional model classes. Spurious rewards remain effective on general-purpose Qwen2.5 models but generally fail to yield any gains on other model families. The performance improvements on non-Qwen2.5 models are substantially smaller compared to those observed in the Qwen2.5 family.

Practical warning ⚠: Proposed RLVR reward signals should be tested on diverse models! Many recent methods on RLVR for reasoning draw their conclusions primarily or exclusively from gains shown on Qwen models (non-exhaustively, Zuo et al. (2025); Zhao et al. (2025c); Wang et al. (2024); Xie et al. (2025); Hu et al. (2025a); Zhang et al. (2025)). As a case study, we experimented with recent work on (1) test time training (Zuo et al., 2025) and (2) one-shot RL (Wang et al., 2025). As shown in Figure 4, these methods exhibits a similar pattern to our results above: the proposed training signals yield strong improvements on Qwen2.5-Math or Qwen2.5 models, matching the performance of the ground truth rewards. However, these same signals often fail to yield performance gains on other model families. See Appendix A.6 for details of the setup. Our findings suggest that existing Qwen-centric RLVR research should possibly be further validated on non-Qwen models.

4 What Makes RLVR with Spurious Rewards Work?

Previously, we demonstrated that different models can exhibit markedly different outcomes when trained with the same reward function. In this section, we investigate *why* such discrepancies occur.

Broadly, we hypothesize that differences in RLVR training outcomes are due to differences in the specific reasoning strategies learned by each model during pretraining. In particular, some strategies may be readily elicited by RLVR, while others may be more difficult to surface or lacking altogether. Below, we identify one such pre-existing strategy—generating code to assist in math reasoning—that Qwen-Math utilizes effectively, and other model families less so. Tracing the prevalence of



(a) TTRL Results (b) One-Shot RL Results

Figure 4: We evaluate two recent weak supervision RL methods—TTRL (Zuo et al., 2025) and One-Shot RL (Wang et al., 2025)—on diverse base models. We find that the proposed training rewards can consistently work on Qwen models. Yet with few exceptions, those same proposed signals often yield no gains on other model families, mirroring the limited generalization observed when training with our own spurious rewards. See Appendix A.6 for setup details. Note that the TTRL accuracies are not directly comparable to our majority-vote reward results, because (a) TTRL trains on (unlabeled) test prompts while we train on a much larger set of distinct train prompts, and (b) TTRL updates labels during training based on the online policy's majority vote, while we assign labels once offline.

code reasoning over the course of RLVR training, we find strong evidence for our hypothesis. We investigate code reasoning *as an illuminating case study*, not as a complete explanation: we observe other behaviors that are also elicited easily and often correlate with performance. We briefly discuss another such behavior, generation without repetition, in Appendix E.

4.1 Different Models Exhibit Pre-existing Discrepancies in Reasoning Strategies

We begin with a case study: to understand the discrepancy in behaviors between Qwen2.5-Math-7B and OLMo2-7B before RL training, we assess the reasoning traces generated by both models in response to the MATH-500 test set. We observe a striking difference: Qwen2.5-Math-7B frequently generates Python code to assist its thinking process (65.0% of all responses), despite having no access to a code execution environment; we denote this behavior *code reasoning*. A truncated example is shown in Figure 5. We provide more examples and qualitative analysis on Qwen2.5-Math-7B's code reasoning behaviors in Appendix F. In all cases, Qwen2.5-Math-7B generates Python code to support its reasoning process, which we find to be a common pattern in the model generation traces. On the other hand, we did not detect comparable code-assisted reasoning patterns in OLMo2-7B.

Further analysis hints at the origins of this reasoning behavior: Qwen2.5-Math-7B maintains accuracy when faced with numerical variations in questions from common math benchmarks—correctly predicting answers when different numbers are substituted in the original problems, echoing existing literature (Huang et al., 2025). Furthermore, the model can often produce complex numerical answers with high precision when predicting code outputs, as shown in Figure 18, 17. However, when those questions are reformulated with an alternative narrative, the model stops utilizing code reasoning approaches, as shown in Figures 23, 24, 25. Hence, we conjecture that Qwen2.5-Math-7B has seen many code-assisted math reasoning traces during pre-training.

Notably, Qwen2.5-Math-7B's use of code is not merely superficial—code use is strongly predictive of answer correctness. As shown in Table 1, Qwen2.5-Math-7B achieves significantly higher accuracy on MATH prompts when code reasoning is used instead of natural language reasoning alone.Qwen2.5-Math-1.5B, pre-trained on the same corpus as Qwen2.5-Math-7B, exhibits similarly effective code reasoning. This pattern is not observed in other models, for which we categorize as "No-Code" or "Bad-Code." *No-Code* models—Llama models, Qwen2.5-1.5B, and OLMo2-7B—do not generate code at all, and consequently do not benefit from this reasoning strategy; *Bad-Code* models frequently attempt to use code reasoning—OLMo2-7B-SFT (98.0% of the time) and Qwen2.5-7B (92.2% of the time)—but code generation correlates with *worse* performance for these models. Thus, *effective* code reasoning is a unique pre-existing capability of the Qwen2.5-Math models before RLVR training.

MATH Question:

What is the distance, in units, between the points $(2, -6)$ and $(-4, 3)$? Express your answer in simplest radical form.

Qwen2.5-Math-7B Solution (correct):

To find the distance between two points (x_1, y_1) and (x_2, y_2) in a Cartesian plane...
Let's break this down step-by-step and compute the result using Python.

```python
import math
...
# Calculate the distance using the distance formula
distance = math.sqrt(dx**2 + dy**2)
print(distance)
```

output: 10.816653826391969
...
Thus, the final answer is: $\boxed{3\sqrt{13}}$

Figure 5: Example of Qwen2.5-Math-7B's code reasoning (see Figure 18 for the complete response). The question is randomly picked from the MATH-500 test set. Note that both the code and the code execution result are autoregressively generated by Qwen2.5-Math-7B. **No external code interpreter was provided to the model.**

Table 1: We track the percentage of model-generated MATH-500 responses that contain Python code before RL training, as well as their accuracy on (1) responses with code and (2) responses with only natural language. Overall, Qwen2.5-Math models achieve higher performance when using code than when not, while other models do not benefit from code reasoning. The unlisted models (Qwen2.5-1.5B, OLMo2-7B, Llama3.1-8B-Instruct, Llama3.2-3B-Instruct) *never* generated code.

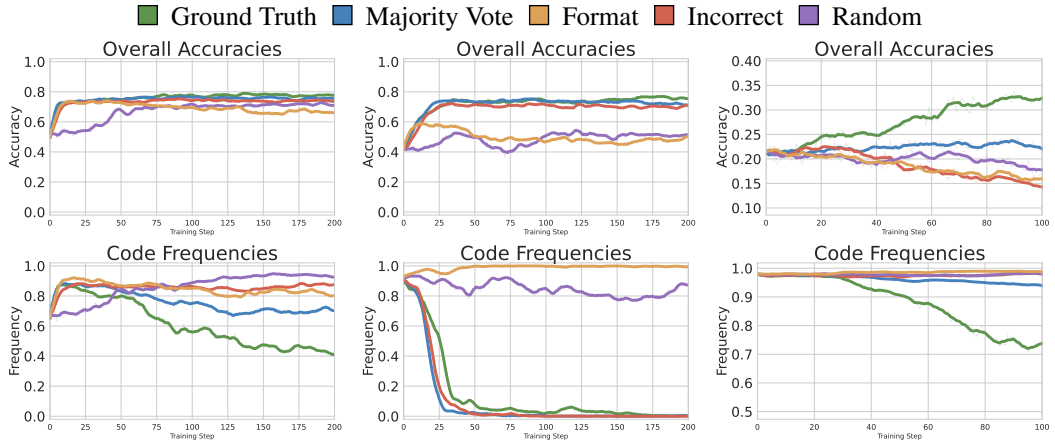
Model	Qwen2.5-Math-7B	Qwen2.5-Math-1.5B	Qwen2.5-7B	OLMo2-7B-SFT
Code Frequency	65.0	53.6	92.2	98.0
Acc. w/ Code	60.9	52.6	39.9	21.0
Acc. w/ Lang	35.0	17.2	61.5	40.0

4.2 RLVR with Spurious Rewards Can Upweight Pre-existing Reasoning Strategies

Motivated by our observations above, we traced changes in the reasoning behavior of models throughout the course of RLVR training across two dimensions: (1) **Accuracy**: the average accuracy of the model on MATH-500, and (2) **Code reasoning frequency**: the percentage of model responses containing the string "`python`". We find that rewards that we employed in the paper, including spurious rewards such as the random and incorrect rewards, gain much of the accuracy on the Qwen2.5-Math and Qwen2.5 through eliciting the correct reasoning strategy.

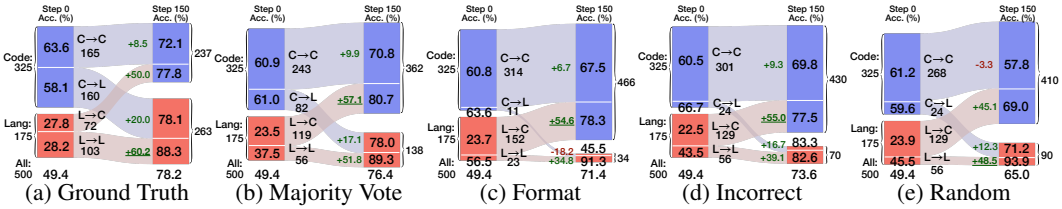
Performance is correlated with code reasoning frequency. As shown in Figure 6, prior to RLVR training, Qwen2.5-Math-7B exhibits a high rate of 65.0% code reasoning solutions, despite not being explicitly prompted to use a code interpreter or being provided with one. After RLVR on all but rewards, the frequency of code reasoning quickly increases to around 90% in the first 15 steps, correlated strongly with the accuracy improvements; random reward shows a more gradual increase in code frequency, but eventually reaches as high as 95.6%. We note that code reasoning frequency also sharply increases with RLVR training on ground truth labels, but gradually drops as the model's natural language reasoning accuracy increases, suggesting that the model is learning real knowledge from RLVR on high-quality, ground truth rewards. For the *Bad-Code* models, we find that *the reduction* of code reasoning frequency is highly correlated to performance gains.

Fine-grained performance by strategy switching. Qwen2.5-Math-7B's accuracy increases by an average of 23.5 absolute points across different training signals. To further break down this gain,



(a) Qwen2.5-Math-7B (b) Qwen2.5-7B (c) OLMo2-7B-SFT

Figure 6: We track the MATH-500 performance (left) and proportion of generated answers that contain Python code blocks (right) of a Qwen2.5-Math-7B model trained with various reward signals. As training with weak (format) or spurious rewards (random, incorrect) progresses, both accuracy and code frequency go up; RLVR with weak or spurious rewards primarily serves to upweight this pre-existing reasoning behavior. Training with ground truth also increases accuracy, but code frequency eventually decreases with continued training, suggesting a possible difference in improvement mechanism from weak or spurious rewards.



(a) Ground Truth (b) Majority Vote (c) Format (d) Incorrect (e) Random

Figure 7: Reasoning strategy switching and fine-grained performance of Qwen2.5-Math-7B on the MATH-500 test set before and after RLVR with different training signals. Blue labels are the problem for which the model uses code reasoning, while red labels indicate reasoning traces using only natural language. Accuracy of each disjoint subset of problems before and after RLVR is shown in the shaded ends, and the size of each subset is shown in the lightly shaded region along with the change in accuracy. For all weak and spurious rewards, the model tends to use more code reasoning after RLVR. There is a small proportion of originally-code-reasoning problems being switched to language reasoning (Code→Lang); a majority of originally-language-reasoning problems convert to code reasoning (Lang→Code), on which we see the most significant performance increase after RLVR.

we track the performance of models trained on each training signal across four disjoint subsets of the test prompts: **(1) Code→Code**: the model uses code reasoning both before and after RLVR; **(2) Code→Lang**: the model initially uses code reasoning but switches to natural language reasoning; **(3) Lang→Code**: the model initially uses natural language reasoning but switches to code reasoning; and **(4) Lang→Lang**: the model uses natural language reasoning both before and after RLVR. Specifically, we focus on two interconnected metrics: frequency and accuracy of each subset. To systematically quantify the contribution of each subset to the performance gain, we define a *Partial Contribution Score*, C_d, for any subset $d \subseteq \mathcal{D}$ of the entire test set $\mathcal{D}$, such that C_d is the ratio between the net increase in the number of correctly answered problems in d divided by the net increase in the number of correctly answered problems in $\mathcal{D}$:

$$C_d = \frac{\sum_{x \in d} \mathbb{I}[\text{correct}(x_t)] - \mathbb{I}[\text{correct}(x_0)]}{\sum_{x \in \mathcal{D}} \mathbb{I}[\text{correct}(x_t)] - \mathbb{I}[\text{correct}(x_0)]}, \quad \text{where } x_0 \text{ and } x_t \text{ are the initial and final answers.}$$

Frequency: The reasoning strategy switching pattern of Qwen2.5-Math-7B is illustrated in Figure 7. For all weak and spurious rewards, the model tends to use more code reasoning after RLVR. There is a small proportion of originally-code-reasoning problems being switched to language reasoning

(Code→Lang), but a majority of originally-language-reasoning problems convert to code reasoning (Lang→Code). Note that ground truth reward does not conform to the pattern observed in spurious rewards. Similarly, we track the fine-grained performance change in each of the 4 subsets for *Bad-Code* and *No-Code* models as well. For *Bad-Code* models (Qwen2.5-7B and OLMo2-7B-SFT), we find models are more easily steerable *away* from bad code reasoning with meaningful rewards (ground truth). As shown in Figures 6b and 6c, (bad) code reasoning decreases with ground truth, majority vote, and incorrect rewards for Qwen2.5-7B, but only with ground truth and majority vote rewards for OLMo2-7B-SFT. For *No-Code* models, RLVR on any training signal fails to elicit meaningful changes in reasoning strategy, as this capability is likely not learned by the model during pre-training.

Table 2: Partial contribution to the overall performance gain averaged over rewards that successfully steered the model's reasoning strategy (Figure 6).

Model	**Qwen2.5-Math-7B**	**Qwen2.5-Math-1.5B**	**Qwen2.5-7B**
Avg. Total Gain	↑ 23.5%	↑ 28.5%	↑ 30.6%
$\mathbf{C}_{\textbf{Code}\rightarrow\textbf{Code}}$	11.6%	2.8%	5.0%
$\mathbf{C}_{\textbf{Code}\rightarrow\textbf{Lang}}$	8.6%	2.0%	**93.9%**
$\mathbf{C}_{\textbf{Lang}\rightarrow\textbf{Code}}$	**58.3%**	**78.7%**	0.0%
$\mathbf{C}_{\textbf{Lang}\rightarrow\textbf{Lang}}$	21.4%	16.5%	5.9%

Accuracy: From Figure 7, there is a drastic increase in accuracy in the Lang→Code subset after RLVR across all training signals. This is reflected in Table 2, which shows that 58.3% of the performance gain of Qwen2.5-Math-7B is from this subset. Similarly in Qwen2.5-Math-1.5B, switching from natural language reasoning to code reasoning contribute to 78.7% of the performance gain. For the *Bad-Code* models, Code→Lang contributes to 93.9% of the performance gain of Qwen2.5-7B. This is intuitive, as the model has a higher langauge reasoning accuracy than code reasoning accuracy, RLVR training essentially encourages the model to use the reasoning strategy that it is better at. For *No-Code* models, since there is no code reasoning before or after RLVR, all performance gains (or losses) are from the **Lang→Lang** subset. These results suggest that much of the accuracy gain from RLVR on these spurious rewards in Qwen2.5-Math and Qwen2.5 is simply from eliciting the right reasoning strategy from the model.

4.3 Intervening Explicitly on Code Reasoning Frequency

We have shown observationally that code reasoning frequency increases during RLVR, which correlates with increased test performance. Here, we investigate the causal impact of code reasoning by explicitly inducing it to occur more or less frequently. If our hypothesis is correct—that increased code reasoning is one primary driver of Qwen2.5-Math-7B's performance gains when training with spurious rewards—these interventions should produce corresponding performance gains or declines.

Inducing code reasoning significantly improves Qwen2.5-Math models' performance, and generally degrades other models. We deliberately induce more frequent code reasoning behaviors via (1) prompting and (2) RLVR training. To induce with prompting, we force the models to begin their responses with "Let's solve this using Python." As shown in Table 3, the MATH-500 accuracy of Qwen2.5-Math-1.5B, Qwen2.5-Math-7B, and Qwen2.5-1.5B improved by 25.6%, 11.8%, and 10.2%, respectively. However, we see performance degradation for other models such as Llama and OLMo2. This drop is consistent with our earlier observation that the other models do not exhibit effective code reasoning behaviors (Section 4.1).

To induce increased coding reasoning frequency with RLVR, we train by assigning a positive reward if and only if a response contains the string "`python`", which we term as a *Python reward*. This intervention is effective; we find that Qwen2.5-Math-7B generated code reasoning solutions in > 99% of its answers after just 20 training steps. We show MATH-500 performance during training in Figure 8. Overall, we observe performance gains specifically in the Qwen2.5-Math models. Other models demonstrated limited improvement. For Qwen2.5-Math, we observe that inducing code reasoning with RLVR matches or exceeds the performance gains from inducing via prompting.

Inhibiting code reasoning during RLVR with spurious rewards can reduce gains on Qwen2.5-Math-7B, but increase gains on other models. We hypothesized that code reasoning is part of the source of weak and spurious rewards gains. Contrapositively, *penalizing* code frequency could

potentially reduce the gains of these rewards in Qwen2.5-Math. To test this, we designed compound rewards that intersect each weak or spurious reward with a *no Python reward*, so that a response is

Table 3: Model performance on MATH-500 after augmenting the prompt to incentivize code reasoning. In this experiment, we force the model's first generated sentence to be "Let's solve this using Python." When applied to Qwen2.5-Math models, which have strong code reasoning priors, our "code-forcing" prompting strategy results in significantly increased test accuracy.

Model	Original	Prompting	Abs. Diff.
Qwen2.5-Math-1.5B	34.8%	60.4%	+25.6%
Qwen2.5-Math-7B	52.6%	64.4%	+11.8%
Qwen2.5-1.5B	2.8%	13.0%	+10.2%
Qwen2.5-7B	42.8%	22.2%	–20.6%
Llama3.2-3B-Instruct	36.6%	8.2%	–28.4%
Llama3.1-8B-Instruct	38.4%	15.2%	–23.2%
OLMo2-7B	10.2%	7.8%	–2.4%
OLMo2-7B-SFT	20.4%	18.6%	–1.8%

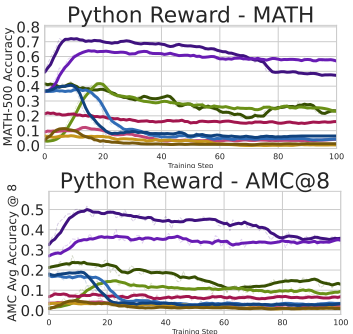
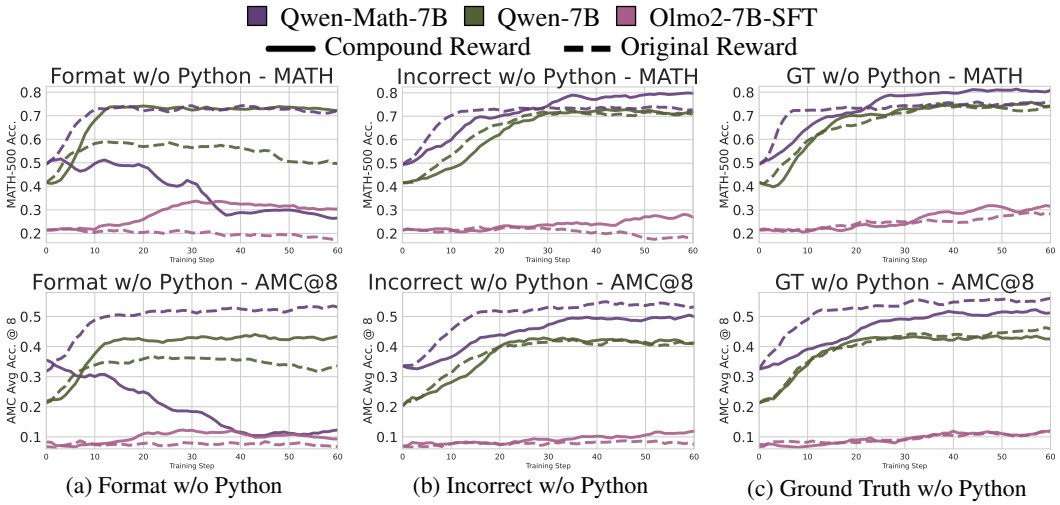


Figure 8: Performance when using our Python reward to explicitly encourage the model to perform code reasoning. This improves performance in Qwen2.5-Math but not other models. Qwen2.5-Math-7B starts to generate > 99% code reasoning in 20 training steps.



(a) Format w/o Python (b) Incorrect w/o Python (c) Ground Truth w/o Python

Figure 9: RLVR with compound rewards that intersect (i) our original rewards with a (ii) *no Python reward* that only rewards responses without Python code. We defer corroborating results on AIME and more models to Appendix D.

rewarded if and only if (1) it satisfies the original spurious reward condition and (2) it does not contain the string "`python`". Consistent with our hypothesis, format rewards cease to improve Qwen2.5-Math-7B when paired with the no code reward (Figure 9c). The spurious incorrect compound reward matches the original incorrect reward on MATH-500; on more difficult benchmarks such as AMC and AIME, gains also persist but are reduced (Figures 9b, 15). Thus, while ablating code reasoning does hurt the incorrect reward as predicted, we posit that other beneficial behaviors (e.g., reduced repetition, see Appendix E) may still be superficially elicited. In addition, ground-truth rewards still yield gains, suggesting benefits beyond eliciting code reasoning (consistent with the code-frequency trends in Figure 6).

Intriguingly, for *Bad-Code* models Qwen2.5-7B and OLMo2-7B-SFT, compound rewards often *outperform* the originals. Most strikingly, OLMo2-7B-SFT—which degrades under standalone format or incorrect rewards—gains +8.9 and +5.5 points, respectively, once the no-code reward is added. We hypothesize that this is because Qwen2.5-7B and OLMo2-7B-SFT exhibit weak code reasoning

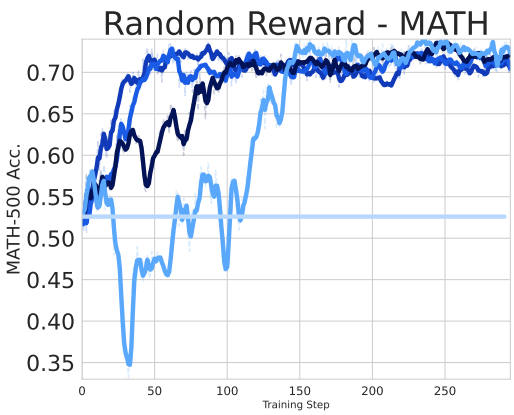


Figure 10: We train Qwen2.5-Math-7B using GRPO with random rewards of different probability $\gamma \in \{0.7, 0.5, 0.3, 0.001, 0\}$. We note that with the exception of $\gamma = 0$, which leads the model to not perform any learning, all other probability configuration successfully leads the model to achieve a significant performance gain after some period of random walk. Furthermore, once in the high-accuracy regime, the training procedure maintain model's performance.

Random $\gamma = 0.7$ Random $\gamma = 0.5$
Random $\gamma = 0.3$ Random $\gamma = 0.001$
Random $\gamma = 0$

before RLVR training, so compound rewards explicitly downweight a behavior that is suboptimal for these models.

4.4 The Curious Case of Incorrect and Random Rewards

In this section, we discuss our hypothesis on the source of RLVR training signals with incorrect or random rewards.

4.4.1 Incorrect Rewards

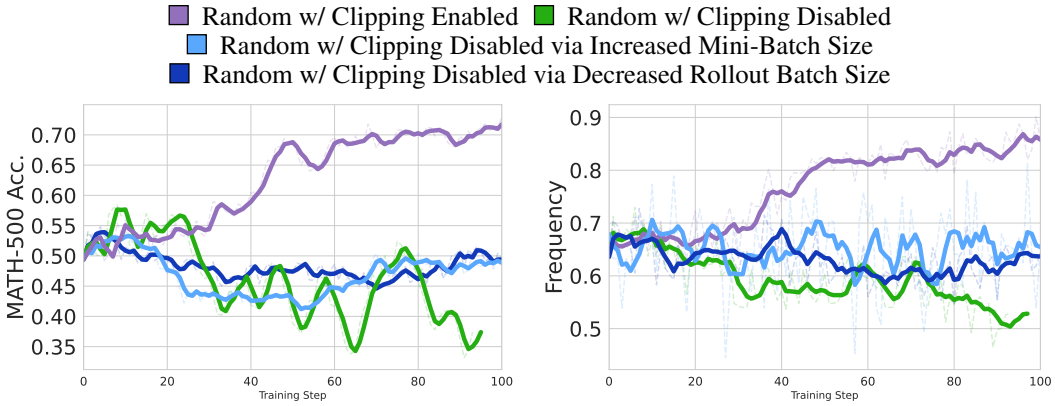
We hypothesize two mechanisms that allows incorrect reward to provide effective training signals that elicits the right reasoning behavior. First, many incorrect labels remain close to ground truth values, allowing the model to receive positive reinforcement for largely correct reasoning traces. Second, incorrect labels may function similarly to format rewards: the model cannot assign positive rewards without successfully extracting and evaluating the generated answer, which requires some degree of correct reasoning.

4.4.2 Random Rewards

The source of training signals in random-reward RLVR is not immediately apparent, given that the rewarding process randomly rewards and penalizes all rollouts, both good and bad. In this section, we first show our observation with random rewards holds for different probabilities. Then , we discuss (1) the source of training signals, that is, why the expectation of the policy gradient is not zero, and (2) the potential reason that useful patterns are up-weighed after training with random rewards. Finally, we verify our assumption with experiments.

Random rewards with varying probabilities consistently improve performance. We train Qwen2.5-Math-7B using GRPO with random rewards assigned by $\texttt{Bernoulli}(\gamma)$ variables, where $\gamma \in \{0.7, 0.5, 0.3, 0.001, 0\}$. Each response receives reward 1 with probability γ and 0 otherwise. Figure 10 shows that all non-zero probability configurations successfully lead to significant performance gains after an initial period of exploration, yielding comparable performance to ground truth rewards. $\gamma = 0$ yields no improvement as expected, since constant rewards provide no learning signal. The convergence speed varies with γ, but all configurations eventually reach similar high-performance regimes, with accuracy improvements of 15-20 percentage points on MATH-500.

GRPO clipping bias can induce random reward training signals. With random rewards, any response is equally likely to be rewarded and penalized. As we derive in Appendix B.1.1, the expectation of the advantage, which decides the direction and magnitude of the gradient updates in the model, is zero. Despite this zero-expected advantage property, the expected gradient in the GRPO loss is nonzero due to the clipping mechanism in the loss—we present a detailed derivation in Appendix B.1. Following Yu et al. (2025)'s observation that clipping bias suppresses exploration while favoring exploitation in RLVR with ground-truth labels, we hypothesize that this bias exhibits



(a) Performance on MATH-500 (b) Frequency of Code Reasoning

Figure 11: Performance and code reasoning frequencies when ablating the clipping term in GRPO for Qwen2.5-Math-7B. We remove the clipping term through three methods: (i) disabling clipping in the implementation, (ii) increasing mini-batch size to match rollout size and (iii) reducing rollout size to maintain equivalent conditions. Both (ii) and (iii) ensures that there is only 1 gradient update per rollout phase, so $\pi_\theta = \pi_{\text{old}}$, directly avoiding any clipping. From our observation, training with random rewards with clipping produces an increase in code reasoning patterns and improved performance. Other runs without clipping fail to replicate the same behavior within 100 steps. In Appendix B.2, we show that no-clipping runs with different random seeds are highly stochastic, which can randomly achieve performance improvement or decrease by chance.

similar behavior under random rewards by reinforcing the model's high-prior model behavior. We detail this hypothesis in Appendix B.1.

To test this hypothesis, we ablate the effect by circumventing the clipping bias throughout training. To achieve this, we either (a) directly turn off the clipping term in the loss calculation or (b) adjust training and rollout batch sizes to ensure $\pi_\theta = \pi_{\text{old}}$, thus preventing triggering clipping constraints. As shown in Figure 11, under standard GRPO with clipping (purple line), random rewards increase code reasoning behavior under standard clipping, effectively concentrating the model on its existing reasoning pattern distribution. This concentration effect vanishes when clipping is disabled across different experimental conditions (other lines). This increase in code reasoning correlates with improved performance. Random rewards produce a $\sim 21\%$ performance gain. However, when clipping effects are eliminated, random rewards yield no robust improvement. In Appendix B.2, we find that training without clipping is very unstable and there is a chance that the model can achieve high performance across multiple runs with different random seeds. We defer an in-depth analysis on the exact mechanism to future work.

Our findings reveal that GRPO's clipping mechanism provides a meaningful training signal even from pure noisy rewards. We conjecture that the apparent "training signal" in random reward training is an artifact of the optimization algorithm's bias toward exploiting existing priors learned in pretraining. There might exist more factors beyond clipping that can contribute to model improvement with random rewards, and we defer the study on the exact mechanism to future work.

5 Related Work

Reinforcement Learning for Language Models. The development of language model capabilities has been significantly advanced through reinforcement learning approaches. RLHF has become a standard technique for aligning models with human preferences (Ouyang et al., 2022; Bai et al., 2022), while RLVR has proven effective for tasks with deterministic answers (DeepSeek-Math, 2024; Gao et al., 2024; Wen et al., 2025; Song et al., 2025; Team et al., 2025; Lambert et al., 2024; Zeng et al., 2025; Luo et al., 2025a,b; Liu et al., 2025; Fatemi et al., 2025; He et al., 2025; Team, 2025; Wang et al., 2025; Zhao et al., 2025a). These methods traditionally rely on accurate supervision signals, either through human feedback or verifiable rewards. Recent work has explored reducing the dependence on human annotations through AI feedback mechanisms (Bai et al., 2022) and

through training dynamics analysis (Zhao et al., 2025b), which supports the finding that RL primarily amplifies behaviors or capabilities already buried in the pretrained models (Liu et al., 2025; Yue et al., 2025; Gandhi et al., 2025).

Unsupervised Reinforcement Learning. Several approaches have explored unsupervised reinforcement learning. Prasad et al. (2024) introduced Self-Consistency Preference Optimization (ScPO), which trains models to prefer consistent answers over inconsistent ones on unsupervised problems. Similarly, Test-Time Reinforcement Learning (TTRL) (Zuo et al., 2025) leverages majority voting across sampled outputs to estimate pseudo-rewards, demonstrating significant performance improvements on mathematical reasoning tasks. EMPO (Zhang et al., 2025) adapts PPO (Schulman et al., 2017) or GRPO (Shao et al., 2024) for unsupervised RLVR by calculating the rewards based on minimizing the entropy of queries in the semantic space. These approaches suggest that internal model consistency can serve as an effective proxy for correctness, though they do not systematically investigate how different kinds of training rewards affect various model families during RLVR.

6 Discussion

Our research demonstrates that RLVR with weak or spurious rewards (format-only, random, and incorrect) improves reasoning in Qwen2.5-Math models largely by amplifying existing reasoning patterns. As one example, we find that RLVR encourages more frequent code reasoning—a capability already present in the pretrained model (e.g., Qwen2.5-Math-7B) that correlates with higher accuracy. Our observational experiments confirm that both code usage frequency and test accuracy increase during RLVR training across all reward settings uniquely for Qwen2.5-Math models. As further validation, we show that directly inducing code reasoning results in strong performance gains.

Our findings have three main implications: base model pretraining significantly affects RLVR outcomes; even corrupted or spurious supervision can enhance reasoning when it triggers useful existing behaviors; and effects observed in one model family may not generalize to others. Our work highlights the importance of testing across multiple models with differing pretraining distributions when evaluating reinforcement learning techniques.

Acknowledgments

We thank Hamish Ivison and Trung Vu for insightful discussions. This research was developed in part with funding from the Defense Advanced Research Projects Agency's (DARPA) SciFy program (Agreement No. HR00112520300). The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. This work was supported by the Singapore National Research Foundation and the National AI Group in the Singapore Ministry of Digital Development and Information under the AI Visiting Professorship Programme (award number AIVP-2024-001), and by the AI2050 program at Schmidt Sciences. This work was supported by NSF grants 2112471 and 2134012. SG is supported by the NSF GRFP. SSD acknowledges the support of NSF IIS-2110170, NSF DMS-2134106, NSF CCF-2212261, NSF IIS-2143493, NSF CCF-2019844, NSF IIS-2229881, and the Sloan Research Fellowship.

References

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

DeepSeek-Math. Deepseek-r1-zero: Advancing mathematical reasoning through step-by-step exploration. *arXiv preprint arXiv:2404.01140*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony S. Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aur'elien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cris

tian Cantón Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab A. AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriele Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guanglong Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Ju-Qing Jia, Kalyan Vasuden Alwala, K. Upasani, Kate Plawiak, Keqian Li, Ken-591 neth Heafield, Kevin R. Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuen ley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melissa Hall Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Niko lay Bashlykov, Nikolay Bogoychev, Niladri S. Chatterji, Olivier Duchenne, Onur cCelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasić, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Ro main Sauvestre, Ron nie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sa hana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Chandra Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Vir ginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whit ney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yiqian Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zhengxu Yan, Zhengxing Chen, Zoe Papakipos, Aaditya K. Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adi Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Po-Yao (Bernie) Huang, Beth Loyd, Beto de Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Shang-Wen Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzm'an, Frank J. Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory G. Sizov, Guangyi Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Han Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kaixing(Kai) Wu, U KamHou, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, A Lavender, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie,

Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pe dro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollár, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sung-Bae Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Andrei Poenaru, Vlad T. Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xia Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models. *ArXiv*, abs/2407.21783, 2024. URL https://api.semanticscholar.org/CorpusID:271571434.

Mehdi Fatemi, Banafsheh Rafiee, Mingjie Tang, and Kartik Talamadupula. Concise reasoning via reinforcement learning. *arXiv preprint arXiv:2504.05185*, 2025.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307*, 2025.

Jiaxuan Gao, Shusheng Xu, Wenjie Ye, Weilin Liu, Chuyi He, Wei Fu, Zhiyu Mei, Guangju Wang, and Yi Wu. On designing effective rl reward at training time for llm reasoning. *arXiv preprint arXiv:2410.15115*, 2024.

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

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

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

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025a.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model, 2025b. URL https://arxiv.org/abs/2503.24290.

Kaixuan Huang, Jiacheng Guo, Zihao Li, Xiang Ji, Jiawei Ge, Wenzhe Li, Yingqing Guo, Tianle Cai, Hui Yuan, Runzhe Wang, Yue Wu, Ming Yin, Shange Tang, Yangsibo Huang, Chi Jin, Xinyun Chen, Chiyuan Zhang, and Mengdi Wang. MATH-Perturb: Benchmarking LLMs' math reasoning abilities against hard perturbations. *arXiv preprint arXiv:2502.06453*, 2025.

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. 2024.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13:9, 2024.

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.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Michael Luo, Sijun Tan, Roy Huang, Xiaoxiang Shi, Rachel Xin, Colin Cai, Ameen Patel, Alpay Ariyak, Qingyang Wu, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepcoder: A fully open-source 14b coder at o3-mini level. https://pretty-radio-b75.notion.site/DeepCoder-A-Fully-Open-Source-14B-Coder-at-O3-mini-Level-1cf81902c14680b3bee5eb349a512a51, 2025a. Notion Blog.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a 2025b. Notion Blog.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James Validad Miranda, Jacob Daniel Morrison, Tyler C. Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Chris Wilhelm, Michael Wilson, Luke S. Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hanna Hajishirzi. 2 olmo 2 furious. *ArXiv*, abs/2501.00656, 2024. URL https://api.semanticscholar.org/CorpusID:275213098.

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

Archiki Prasad, Thomas Webb, Kiran Valmeekam, Jay Pujara, and Thomas Wolf. Self-consistency preference optimization. *arXiv preprint arXiv:2411.04109*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. c policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Darsh J Shah, Peter Rushton, Somanshu Singla, Mohit Parmar, Kurt Smith, Yash Vanjani, Ashish Vaswani, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, et al. Rethinking reflection in pre-training. *arXiv preprint arXiv:2504.04022*, 2025.

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

Mingyang Song, Mao Zheng, Zheng Li, Wenjie Yang, Xuan Luo, Yue Pan, and Feng Zhang. Fastcurl: Curriculum reinforcement learning with progressive context extension for efficient training r1-like reasoning models. *arXiv preprint arXiv:2503.17287*, 2025.

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

OpenThoughts Team. Open Thoughts. https://open-thoughts.ai, January 2025.

Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M Ni, et al. Openr: An open source framework for advanced reasoning with large language models. *arXiv preprint arXiv:2410.09671*, 2024.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Lucas Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, et al. Reinforcement learning for reasoning in large language models with one training example. *arXiv preprint arXiv:2504.20571*, 2025.

Liang Wen, Yunke Cai, Fenrui Xiao, Xin He, Qi An, Zhenyu Duan, Yimin Du, Junchen Liu, Lifu Tang, Xiaowei Lv, et al. Light-r1: Curriculum sft, dpo and rl for long cot from scratch and beyond. *arXiv preprint arXiv:2503.10460*, 2025.

Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2502.14768*, 2025.

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

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, Shanghaoran Quan, and Zekun Wang. Qwen2.5 technical report. *ArXiv*, abs/2412.15115, 2024b. URL https://api.semanticscholar.org/CorpusID:274859421.

Wenkai Yang, Shuming Ma, Yankai Lin, and Furu Wei. Towards thinking-optimal scaling of test-time compute for llm reasoning. *arXiv preprint arXiv:2502.18080*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

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? *arXiv preprint arXiv:2504.13837*, 2025.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*, 2025.

Qingyang Zhang, Haitao Wu, Changqing Zhang, Peilin Zhao, and Yatao Bian. Right question is already half the answer: Fully unsupervised llm reasoning incentivization. *arXiv preprint arXiv:2504.05812*, 2025.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025a.

Rosie Zhao, Alexandru Meterez, Sham Kakade, Cengiz Pehlevan, Samy Jelassi, and Eran Malach. Echo chamber: Rl post-training amplifies behaviors learned in pretraining. *arXiv preprint arXiv:2504.07912*, 2025b.

Xuandong Zhao, Zhewei Kang, Aosong Feng, Sergey Levine, and Dawn Song. Learning to reason without external rewards. *arXiv preprint arXiv:2505.19590*, 2025c.

Yuxin Zuo, Kaiyan Zhang, Shang Qu, Li Sheng, Xuekai Zhu, Biqing Qi, Youbang Sun, Ganqu Cui, Ning Ding, and Bowen Zhou. Ttrl: Test-time reinforcement learning. *arXiv preprint arXiv:2504.16084*, 2025.

Appendix

A Experimental Setup

A.1 Preliminaries on GRPO

Denote the input prompt as x and the corresponding model rollouts as y. Denote the t-th token in the i-th rollout y as y_t, where $1 \leq t \leq |y|$. GRPO loss, as introduced by (Shao et al., 2024), contains the following components: For each prompt x, we compute the group-wise mean $\bar{r}_x$ and standard deviation σ_x. The normalized *group-relative advantage* is $\hat{A}(x, y) = \frac{r(x,y)-\bar{r}_x}{\sigma_x}$. Let π_{old} be the behavior policy that produced the trajectories and π_{ref} be a frozen reference policy (for example, the initial supervised model). We denote the token-level importance ratio by $\rho_t(y;\theta) = \frac{\pi_\theta(y_t|x,y_{<t})}{\pi_{\text{old}}(y_t|x,y_{<t})}$, where t is the token index. With PPO-style clipping threshold ϵ_c, KL-penalty weight λ, the surrogate objective maximized is

$$
\begin{aligned}
L(\theta) =& \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_{\text{old}}(\cdot|x)} \left[\sum_{t=1}^{|y|} \min\Big(\rho_t(y;\theta)\, \hat{A}(x,y), \text{clip}\big(\rho_t(y;\theta), 1-\epsilon_c, 1+\epsilon_c\big)\, \hat{A}(x,y) \Big) \right] \\
& - \lambda\, \mathbb{E}_{x\sim\mathcal{D}} \Big[\text{KL}\big(\pi_\theta \| \pi_{\text{ref}}\big) \Big].
\end{aligned} \tag{1}
$$

The KL regularization is typically adopted to ensure our model does not deviate too far from the frozen reference model. However, since we focus primarily on verifiable rewards and investigate the signals produced by these rewards, we assume that distribution shift is of lesser concern following (Liu et al., 2025). Moreover, KL regularization adds confounding factors to our analysis. Recent work has also shown that removing the KL term leads to better performance (Hu et al., 2025b). Therefore, we set $\lambda = 0$ in our work.

A.2 Datasets and Models

We conduct our experiments on three canonical mathematical reasoning benchmarks:

- **MATH-500** (Hendrycks et al., 2021; Lightman et al., 2023): A standardized subset of the MATH dataset focusing on advanced mathematical reasoning, including problems from algebra, calculus, and number theory.
- **AMC** (Li et al., 2024): The American Mathematics Competition dataset contains diverse mathematical problems ranging from algebra to combinatorics and geometry. For this benchmark, we report model's average performance over 8 trials (avg@8).
- **AIME** (Li et al., 2024): The American Invitational Mathematics Examination dataset consists of challenging high-school level mathematical problems requiring multi-step reasoning. We include AIME questions from 2024 and 2025 for evaluation. For this benchmark, we report model's average performance over 8 trials (avg@8).

For our initial experiments, we primarily utilize the Qwen2.5-Math-7B model, a 7 billion parameter language model specifically tuned for mathematical reasoning tasks. We select this model (1) for its strong baseline performance on mathematical problems while remaining computationally efficient for multiple experimental iterations and (2) because it is frequently used in prior work (Zuo et al., 2025; Wang et al., 2025).

A.3 Training Configuration

Unless otherwise specified, we train each model on 8 GPUs with a constant learning rate of 5e-7, a mini batch size (number of rollouts seen before a gradient update) of 128, and a rollout batch size (number of prompts we rollout at the same time) of 64. For each prompt, we collect 16 rollouts to compute advantages for GRPO update. We use a sampling temperature $\tau = 1$. We do not apply KL divergence loss or entropy loss in our training.

A.4 Computation Resource

Each RLVR run in our experiment takes approximately 24 hours on 8 A100s.

A.5 Visualization

In all plots, we smooth the metric of interest across ten training steps, and overlay the raw curves as transparent dashed lines. We do not smooth the step 0 performance, which corresponds to the performance of the base model before training. We report the smoothed value at the final training step as the final performance.

A.6 Experimental Setups for TTRL and One-Shot RL

TTRL setup. We follow all hyperparameters from the original TTRL paper (Zuo et al., 2025), and train all models using their publicly released implementation. Due to compute constraints, we do not extensively sweep hyperparameters on the new base models we consider.

Differences in our One-Shot RL setup from Wang et al. (2025). We note that for the one-shot RL settings in our paper, we do not apply entropy loss for more consistent setups with other experiments and better stability. However, Wang et al. (2025) shows that the entropy loss may further improve the performance of one-shot RL and post-saturation generalization. We use the same training example π_1 (defined in Wang et al. (2025)) for one-shot RLVR experiments on all models as in Wang et al. (2025). Their work discusses that different models should possibly select different examples for more performant one-shot RLVR training, but use π_1 in all experiments due to high cost of trying different possible 1-shot examples.

B The Curious Case of Random Rewards

In this section, we study a special case of our spurious rewards, random rewards, where the reward is randomly assigned independently of the model rollouts. We provide detailed gradient derivation in this special case and discuss our hypothesis on one of the potential sources of training signals with random rewards—the clipping bias in GRPO update.

Recap of notations. Recall that we denote the input prompt as x and the corresponding model rollouts as $\{y^{(1)}, \cdots, y^{(G)}\}$, where $G \in \mathbb{N}^+$ is the rollout size. In addition, we denote the t-th token in the i-th rollout $y^{(i)}$ as $y_t^{(i)}$, where $t \in \mathbb{N}^+$ and $1 \leq t \leq |y^{(i)}|$. The normalized group-relative advantage in GRPO is $\hat{A}(x, y) = \frac{r(x,y)-\bar{r}_x}{\sigma_x}$, where $\bar{r}_x$ is the group-wise mean and σ_x is the group-wise standard deviation. The token-level importance ratio[2] is $\rho_t(y;\theta) = \frac{\pi_{\theta,x}(y_t)}{\pi_{\text{old},x}(y_t)}$, where t is the token index.

B.1 Conjecture: Clipping Bias Brings Training Signals under Random Rewards

B.1.1 Expected Advantage of Random Rewards

We investigate the expected advantage when the reward signal is assigned by a random $\texttt{Bernoulli}(\gamma)$ variable. For a prompt x and rollouts $\{y^{(i)}\}_{i=1}^G$, the reward of each $y^{(i)}$ is $r(x, y^{(i)}) \sim \texttt{Bernoulli}(\gamma)$, so the expected average reward of x is γ. Assuming $\hat{A}_i = 0$ when $\sigma_x = 0$, the sum of the normalized advantages over G rollouts $\sum_{i=1}^G \hat{A}(x, y^{(i)}) := \sum_{i=1}^G \frac{r(x,y^{(i)})-\bar{r}_x}{\sigma_x} = \frac{\sum_{i=1}^G r(x,y^{(i)})-G\cdot\bar{r}_x}{\sigma_x} = 0$ by construction. Furthermore, for i.i.d. rewards that are independent of the provided samples, such as the random rewards used in our experiments, $\mathbb{E}(\sum_{i=1}^G \hat{A}(x, y^{(i)})) = G \cdot \mathbb{E}(\hat{A}) = 0 \implies \mathbb{E}(\hat{A}) = 0$.

The clipping term in GRPO loss (Equation 1) prevents excessive deviation from the previous policy, stabilizing training. Recent work suggests this term, which operates on the ratio $\rho_t(y;\theta) = \pi_{\theta,x}(y_t)/\pi_{\text{old},x}(y_t)$, introduces bias toward exploitation in the case of ground truth reward (Yu et al., 2025). Here, we analyze the bias in the settings of the random rewards, where each rollout receives an advantage that is independent of the rollout.

We note that the loss is no longer differentiable everywhere with clipping. In practical implementations, e.g., PyTorch, the gradients will be automatically set to 0 when the value is clipped. For

[2]For simplicity, we denote $\pi(y_t|x, y_{<t})$ as $\pi_x(y_t)$, and similarly $\pi_\theta(y_t|x, y_{<t})$ as $\pi_{\theta,x}(y_t)$, $\pi_{\text{old}}(y_t|x, y_{<t})$ as $\pi_{\text{old},x}(y_t)$.

simplicity, we discuss the gradient of the loss function by assuming 0 gradient at the non-differentiable points.

B.1.2 Gradient Derivation of Clipping Bias under Random Rewards

In this section, we derive the training signals induced by the clipping factor, which we name as a "clipping bias". Specifically, we define the clipping bias as the difference in the expected gradients after adding clipping to the GRPO objective[3]:

$$\texttt{Bias}(\nabla_\theta L(\theta)) = \mathbb{E}_{x,y}[\nabla_\theta L(\theta)] - \mathbb{E}_{x,y}[\nabla_\theta L^{\text{unclipped}}(\theta)].$$

As we derived above, $\mathbb{E}(\hat{A}(x, y_j)) = 0$. Assuming a simple loss with no clipping, the expectation of the policy gradient without the clipping term is also trivially zero given that $\hat{A}$ is independent of other variables:

$$\begin{aligned}
&\mathbb{E}_{x,y}[\nabla_\theta L^{\text{unclipped}}(\theta)] \\
=&\mathbb{E}_{x,y}\left[\frac{1}{|y|}\sum_{t=1}^{|y|}\nabla_\theta \rho_t(y;\theta)\,\hat{A}(x,y)\right] \\
=&\mathbb{E}(\hat{A})\mathbb{E}_{x,y}\left[\frac{1}{|y|}\sum_{t=1}^{|y|}\nabla_\theta \rho_t(y;\theta)\right] \\
=&0.
\end{aligned}$$

Therefore, the clipping bias is

$$\texttt{Bias}(\nabla_\theta L(\theta)) = \mathbb{E}[\nabla_\theta L(\theta)] - \mathbb{E}[\nabla_\theta L^{\text{unclipped}}(\theta)] = \mathbb{E}[\nabla_\theta L(\theta)].$$

Next, we compute the exact form of this clipping bias, i.e., the gradient in GRPO with random rewards. Recall from Eq. 1, the surrogate objective that we aim to maximize is:

$$L(\theta) = \min\Big(\rho_t(y;\theta)\,\hat{A}(x,y), \text{clip}\big(\rho_t(y;\theta), 1-\epsilon_c, 1+\epsilon_c\big)\,\hat{A}(x,y)\Big).$$

For simplicity, we further denote $R_\theta = \rho_t(y;\theta) = \frac{\pi_{\theta,x}(y_t)}{\pi_{\text{old},x}(y_t)}$ to be the token-level importance ratio, $C = \text{clip}(R_\theta, 1-\epsilon_c, 1+\epsilon_c)$ to be the clipping term, and $\hat{A}(x,y)$ as $\hat{A}$. So we have $L(\theta) = \min\big(R_\theta \cdot \hat{A}, C \cdot \hat{A}\big)$. We now analyze the gradient of GRPO loss with respect to the policy model parameters θ based on the sign of $\hat{A}$.

As discussed in Section B.1, the loss function is not differentiable everywhere. Therefore, we discuss the gradients with respect to the differentiable regions below and set the gradients to 0 at the non-differentiable points at the end.

Case 1: $\hat{A} \geq 0$. When $\hat{A} \geq 0$, we can directly take $\hat{A}$ out of the $\min$ function. Thus,

$$L(\theta) = \min\big(R_\theta \cdot \hat{A}, C \cdot \hat{A}\big) = \hat{A} \cdot \min(R_\theta, C).$$

Since C is defined as $C = \text{clip}(R_\theta, 1-\epsilon_c, 1+\epsilon_c)$, we have:

- If $R_\theta < 1-\epsilon_c$, then $C = 1-\epsilon_c$. Thus, $\min(R_\theta, C) = R_\theta$.
- If $1-\epsilon_c \leq R_\theta \leq 1+\epsilon_c$, then $C = R_\theta$. Thus, $\min(R_\theta, C) = R_\theta$.
- If $R_\theta > 1+\epsilon_c$, then $C = 1+\epsilon_c$. Thus, $\min(R_\theta, C) = 1+\epsilon_c$.

Combining these, when $\hat{A} > 0$, the objective is:

$$L^+(\theta) = \hat{A} \cdot \begin{cases} R_\theta, & \text{if } R_\theta < 1+\epsilon_c, \\ 1+\epsilon_c, & \text{if } R_\theta > 1+\epsilon_c. \end{cases}$$

The gradient with respect to θ for this case is:

$$\nabla_\theta L^+(\theta) = \hat{A} \cdot \begin{cases} \nabla_\theta R_\theta, & \text{if } R_\theta < 1+\epsilon_c, \\ 0, & \text{if } R_\theta > 1+\epsilon_c. \end{cases}$$

[3]For simplicity, we denote $\mathbb{E}_{x\sim\mathcal{D},y\sim\pi_{\text{old}}(\cdot|x)}$ as $\mathbb{E}_{x,y}$

Case 2: $\hat{A} < 0$**.** When $\hat{A} < 0$, multiplying by $\hat{A}$ flips the $\min$ function. So,

$$L^-(\theta) = \min(R_\theta \cdot \hat{A}, C \cdot \hat{A}) = \hat{A} \cdot \max(R_\theta, C).$$

Applying a similar analysis, when $\hat{A} < 0$, the gradient with respect to θ for this case is:

$$\nabla_\theta L^-(\theta) = \hat{A} \cdot \begin{cases} 0, & \text{if } R_\theta < 1 - \epsilon_c, \\ \nabla_\theta R_\theta, & \text{if } R_\theta > 1 - \epsilon_c. \end{cases}$$

Combining the cases. Together, the gradient is:

$$\nabla_\theta L(\theta) = \hat{A} \cdot \begin{cases} \nabla_\theta R, & \text{if } \hat{A} > 0 \text{ and } R_\theta < 1 + \epsilon_c, \\ 0, & \text{if } \hat{A} > 0 \text{ and } R_\theta > 1 + \epsilon_c, \\ 0, & \text{if } \hat{A} < 0 \text{ and } R_\theta < 1 - \epsilon_c, \\ \nabla_\theta R, & \text{if } \hat{A} < 0 \text{ and } R_\theta < 1 - \epsilon_c. \end{cases}$$

Then, the clipping bias, $\texttt{Bias}(\nabla_\theta L(\theta))$, is

$$\begin{aligned}
&\texttt{Bias}(\nabla_\theta L(\theta)) \\
=&\mathbb{E}_{\hat{A},x,y}\left[\nabla_\theta L(\theta)\right] \\
=&P(\hat{A} > 0) \cdot \mathbb{E}_{\hat{A}>0,x,y}[\nabla_\theta L^+(\theta)] + P(\hat{A} < 0) \cdot \mathbb{E}_{\hat{A}<0,x,y}[\nabla_\theta L^-(\theta)] + 0 \\
=&\mathbb{E}_{x,y}\left[\begin{cases} P(\hat{A} > 0) \cdot \mathbb{E}_{\hat{A}>0}[\hat{A}] \cdot \nabla_\theta R_\theta, & \text{if } R_\theta < 1 + \epsilon_c, \\ 0, & \text{if } R_\theta > 1 + \epsilon_c. \end{cases}\right] \\
&+ \mathbb{E}_{x,y}\left[\begin{cases} 0, & \text{if } R_\theta < 1 - \epsilon_c, \\ P(\hat{A} < 0) \cdot \mathbb{E}_{\hat{A}<0}[\hat{A}] \cdot \nabla_\theta R_\theta, & \text{if } R_\theta > 1 - \epsilon_c. \end{cases}\right] \\
=&\mathbb{E}_{x,y}\left[\begin{cases} P(\hat{A} > 0) \cdot \mathbb{E}_{\hat{A}>0}[\hat{A}] \cdot \nabla_\theta R_\theta, & \text{if } R_\theta < 1 - \epsilon_c, \\ (P(\hat{A} > 0) \cdot \mathbb{E}_{\hat{A}>0}[\hat{A}] + P(\hat{A} < 0) \cdot \mathbb{E}_{\hat{A}<0}[\hat{A}]) \cdot \nabla_\theta R_\theta, & \text{if } 1 - \epsilon_c < R_\theta < 1 + \epsilon_c, \\ P(\hat{A} < 0) \cdot \mathbb{E}_{\hat{A}<0}[\hat{A}] \cdot \nabla_\theta R_\theta, & \text{if } R_\theta > 1 + \epsilon_c. \end{cases}\right]
\end{aligned}$$

By definition, $\hat{A}$ is a normalized distribution and $\mathbb{E}(\hat{A}) = 0$ from Appendix B.1.1, so

$$\mathbb{E}(\hat{A}) = P(\hat{A} > 0)\,\mathbb{E}_{\hat{A}>0}(\hat{A}) + P(\hat{A} < 0)\,\mathbb{E}_{\hat{A}<0}(\hat{A}) = 0, \text{ and}$$

$$P(\hat{A} > 0)\,\mathbb{E}_{\hat{A}>0}(\hat{A}) = -P(\hat{A} < 0)\,\mathbb{E}_{\hat{A}<0}(\hat{A}) \geq 0.$$

We denote $\mu = P(\hat{A} > 0)\,\mathbb{E}_{\hat{A}>0}(\hat{A}) = -P(\hat{A} < 0)\,\mathbb{E}_{\hat{A}<0}(\hat{A}) > 0$. And we substitute $R_\theta = \frac{\pi_{\theta,x}(y_t)}{\pi_{\text{old},x}(y_t)}$ in the conditions. Finally, we set the gradients on non-differentiable points to 0 and have

$$\texttt{Bias}(\nabla_\theta L(\theta)) = \mu \cdot \mathbb{E}_{x,y}\left[\begin{cases} \nabla_\theta R_\theta, & \text{if } \pi_{\theta,x}(y_t) < \pi_{\text{old},x}(y_t) \cdot (1 - \epsilon_c), \\ 0, & \begin{array}{l}\text{if } \pi_{\text{old},x}(y_t) \cdot (1 - \epsilon_c) \leq \pi_{\theta,x}(y_t) \\ \quad \leq \pi_{\text{old},x}(y_t) \cdot (1 + \epsilon_c),\end{array} \\ -\nabla_\theta R_\theta, & \text{if } \pi_{\theta,x}(y_t) > \pi_{\text{old},x}(y_t) \cdot (1 + \epsilon_c). \end{cases}\right]$$

Therefore, we observe that there is a positive gradient if $R_\theta < 1 - \epsilon_c$, and a negative gradient if $R_\theta > 1 + \epsilon_c$, which means the clipping bias discourages the model from leaving the clipping region.

B.1.3 Clipping Creates Asymmetric Updates Towards Model Prior Knowledge

We hypothesize that this gradient bias increases the likelihood of high-probability rollouts, similar to Yu et al. (2025)'s analysis of RLVR with ground-truth labels. We showcase this trend with a simple example.

Consider a case where a token has a high-probability $\pi_{\text{old},x}(y_t) = 0.85$ and an $\epsilon_c = 0.2$ that we adopt in our experiments. Then, the upper threshold from the bias formula becomes $\pi_{\text{old},x}(y_t) \cdot (1 + \epsilon_c) =$

1.02. Since the probability output by the policy model cannot exceed 1, the upper clipping threshold of 1.02 is never reached. Thus, the bias is nonnegative for this token, leading to a net positive gradient on the policy model, which leads to increase in probability on this token.

On the other hand, for a low-probability token where $\pi_{\text{old},x}(y_t) = 0.02$, the policy model receives a negative gradient when $\pi_{\theta,x}(y_t) > \pi_{\text{old},x}(y_t) \cdot (1 + \epsilon_c) = 0.024$; and positive gradient when $\pi_{\theta,x}(y_t) < \pi_{\text{old},x}(y_t) \cdot (1 - \epsilon_c) = 0.016$ from this bias. The low threshold for negative gradient makes penalties more likely than in the high-probability case.

The clipping range width scales linearly with the original probability: higher-probability tokens have wider ranges and face fewer penalties. This asymmetric treatment prevents low-probability samples from receiving substantial upweighting during training, causing the model to concentrate probability mass on its existing distribution.

Why only Qwen-Math models benefit. The clipping bias mechanism operates on all models, but its effectiveness depends on what behaviors get amplified. The bias systematically favors a model's pre-existing high-probability behaviors, but only benefits performance if those behaviors correlate with correctness. For Qwen2.5-Math models, code reasoning occurs in 65% of responses and correlates strongly with correctness (64% accuracy vs 29% without code). Therefore, when clipping bias increases code reasoning frequency to >90%, performance improves substantially. *No-Code models* (Llama, OLMo2-7B) generate no code reasoning, so clipping bias has no beneficial pattern to amplify. On the other hand, *Bad-Code models* (OLMo2-7B-SFT) generate code 98% of the time but with 21% accuracy vs 40% for natural language—amplifying code reasoning hurts performance. Overall, the clipping bias is universal, but its impact on performance depends entirely on whether the model's dominant pre-existing behaviors happen to be effective reasoning strategies.

Implications. Our findings reveal that GRPO's clipping mechanism provides a meaningful training signal even from pure noisy rewards, by systematically favoring the model's pre-existing behavioral patterns. This suggests that the apparent "reward signal" in random reward training is actually an artifact of the optimization algorithm's bias toward exploiting learned priors rather than exploring new behaviors. This mechanism explains why random rewards work for Qwen2.5-Math models (which have strong code reasoning priors) but fail for other model families lacking these pre-existing capabilities. The training algorithm amplifies whatever reasoning patterns already correlate with correctness, regardless of the reward signal's actual informativeness.

B.2 High Stochasticity in RLVR Without Clipping

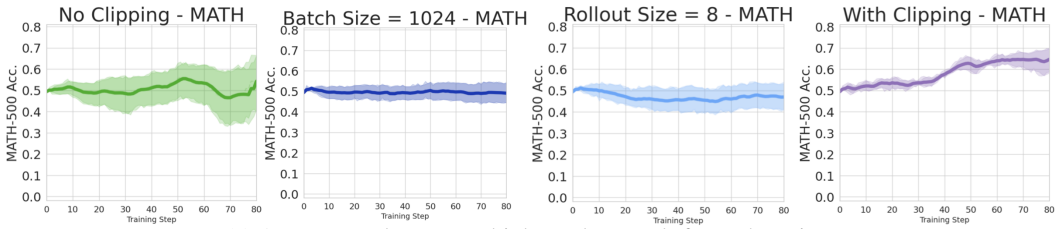
In this section, we empirically show that training without clipping has stochastic training dynamics and can sometimes obtain performance improvement despite the expected gradient being zero. Figure 12 shows training runs on random rewards with clipping disabled using two approaches: (i) removing clipping from the loss implementation, (ii) increasing mini-batch size to match rollout size, and (iii) decreasing the rollout size to match mini-batch size across multiple random seeds.

When clipping is disabled by adjusting batch size, model performance remains stable within a consistent range without converging toward either extreme of the accuracy spectrum. These runs involve 8 times fewer gradient updates due to the batch configuration.

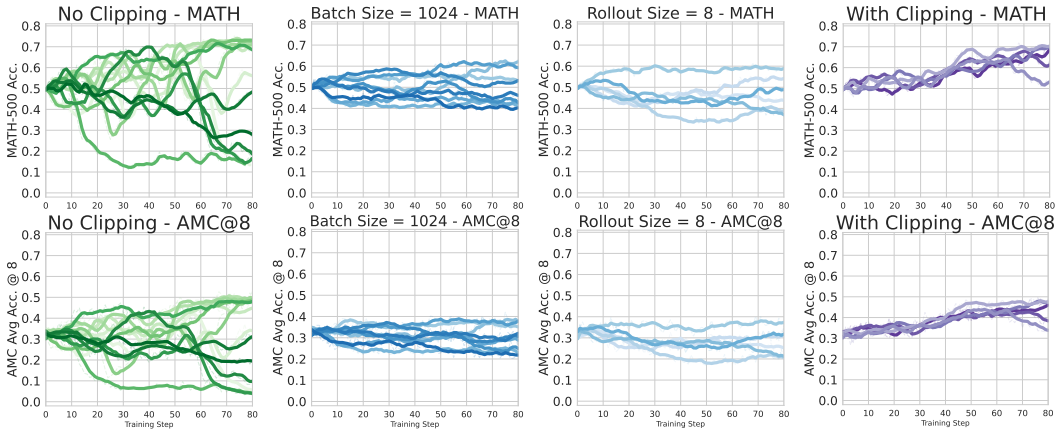
In contrast, removing clipping from the implementation produces extreme stochasticity, occasionally resulting in high-performance convergence across multiple runs. While the exact mechanism remains unclear, we hypothesize that inherent randomness in RLVR training contributes to these observations. The group size $G = 16$ used in GRPO training creates high variance in gradient updates. Combined with the instability of unclipped training, this variance may accidentally reinforce certain reasoning patterns, leading to improved performance in some runs. We leave a systematic analysis of this behavior to future work.

C Additional Results on AIME 2024 and AIME 2025

We present additional results on the AIME benchmarks, which are challenging math Olympiad tests containing significantly harder problems than those in MATH-500 or AMC. We evaluate average@8 accuracy on AIME24 and AIME25 (Li et al., 2024). AIME25 was created after the release date of all



(a) Average results over multiple random seeds for each setting.



(b) Turning off clipping with 8 gradient updates per rollout.

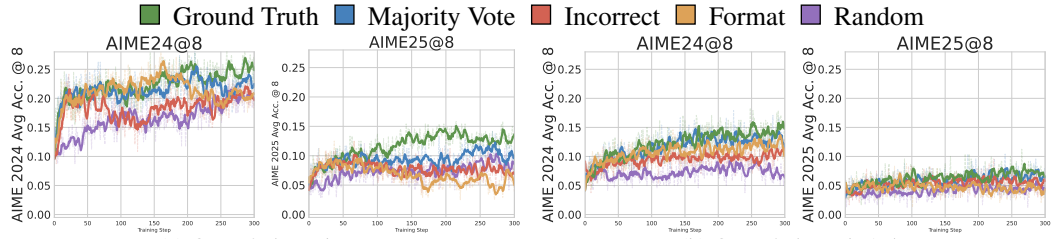
(c) Increasing batch size to ensure only 1 gradient update per rollout.

(d) Decreasing rollout batch size to ensure only 1 gradient update per rollout.

(e) Default GRPO loss with random reward.

Figure 12: RLVR performance on random rewards with disabled clipping across multiple different seeds. Models without clipping show no meaningful performance improvement on average, while models with clipping demonstrate consistent performance gains using random rewards. Disabling clipping in implementation produces high variance in results, occasionally yielding high accuracy scores. Experiments with adjusted batch sizes exhibit greater stability but involve 8 times fewer gradient updates than the disabled clipping experiments.

models considered in our study. Thus, evaluating performance on AIME25 allows us to control the risk that our models' have seen similar problems during web pretraining. We evaluate the trained models from Section 2 and Section 3. We show results on Qwen2.5-Math models in Figure 13 and on the 8 additional models from Section 3 in Figure 14.



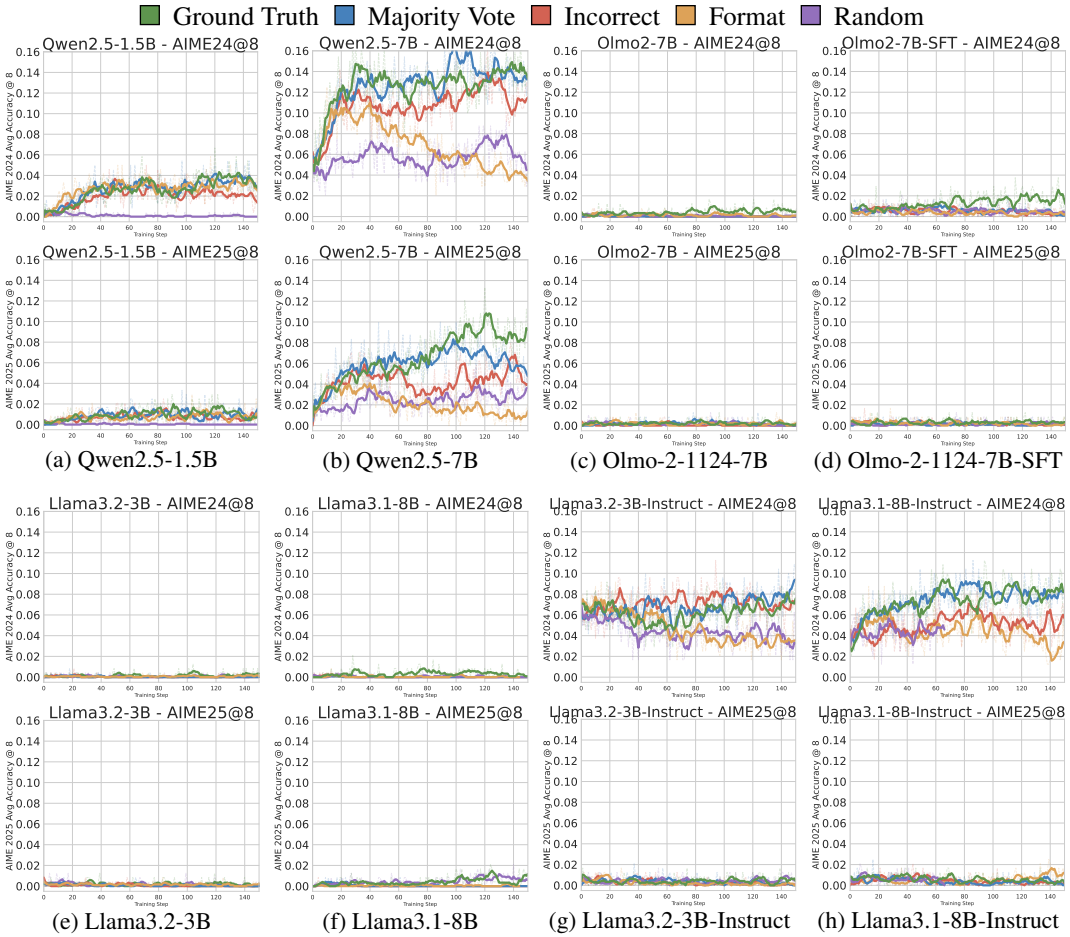
(a) Qwen2.5-Math-7B

(b) Qwen2.5-Math-1.5B

Figure 13: Qwen2.5-Math Model performance on AIME 2024 and AIME 2025.

C.1 Spurious Rewards Yield Significant RLVR Gains on Qwen2.5-Math

As shown in Figure 13, spurious rewards can consistently yield performance gains on Qwen-Math models on AIME24. Intriguingly, we find that any AIME24 gains achievable from training Qwen models with spurious rewards largely vanish when evaluating on AIME 2025. We speculate that AIME25 contains questions that are more out-of-distribution to Qwen's pretrained knowledge; spurious rewards—which largely serve to elicit existing knowledge—hence no longer provide benefit.



(a) Qwen2.5-1.5B (b) Qwen2.5-7B (c) Olmo-2-1124-7B (d) Olmo-2-1124-7B-SFT

(e) Llama3.2-3B (f) Llama3.1-8B (g) Llama3.2-3B-Instruct (h) Llama3.1-8B-Instruct

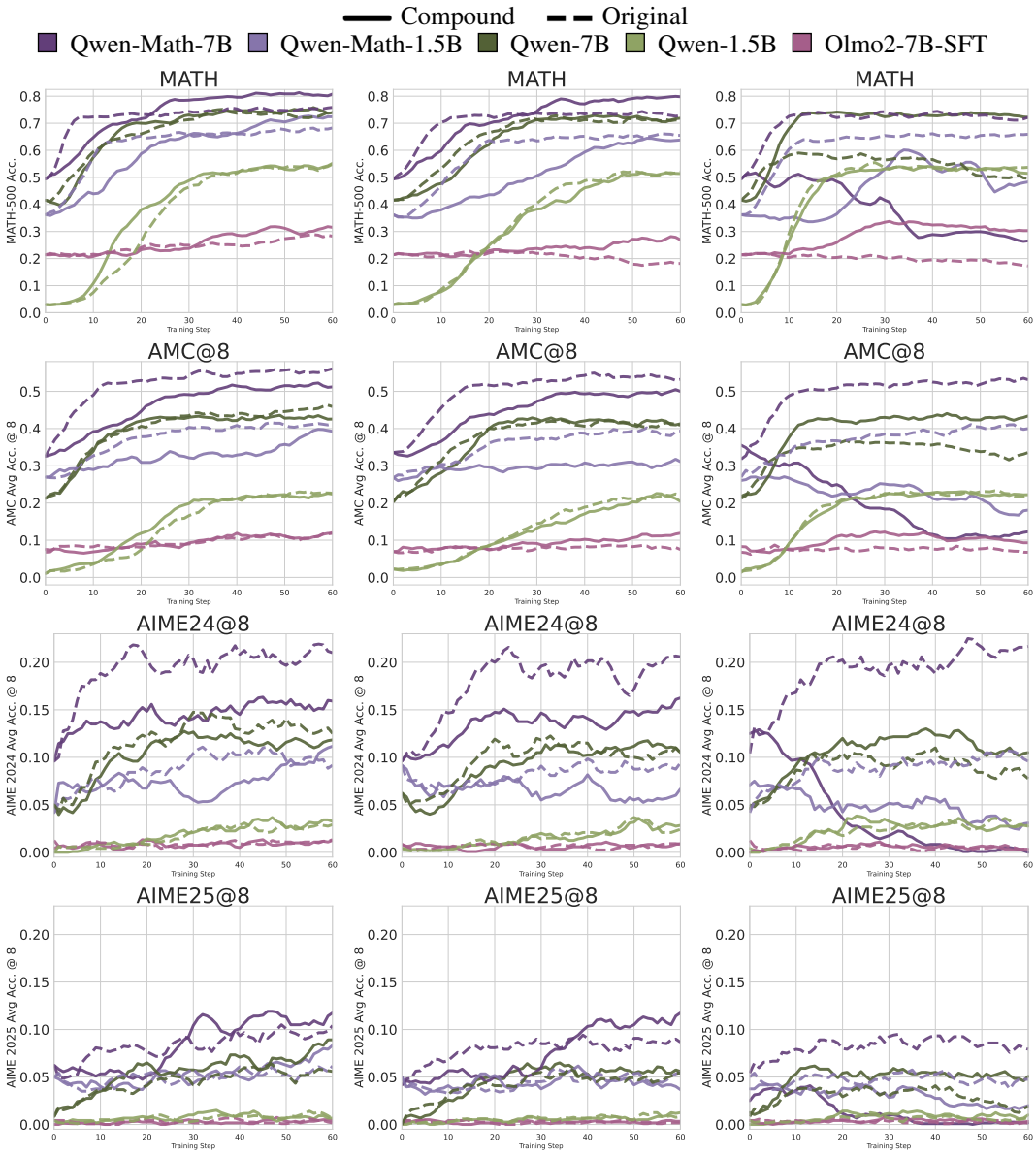
Figure 14: Varying rewards across additional model classes on the AIME 2024 and AIME 2025 benchmarks. Note that AIME 2025 was released after all the models' release dates, serving as a good resource to examine any dataset contamination phenomenon. Weak and spurious rewards (except for random) remain effective on general-purpose Qwen2.5 models but generally fail to yield any gains on other model families. The performance improvements on non-Qwen2.5 models are substantially smaller compared to those observed in the Qwen2.5 family. Note that the AIME benchmarks contain 30 questions each, so small differences in accuracy (less than $\sim$ 2 pp.) may not be significant.

C.2 (Lack of) Generalization to Other Models

Overall, results on other models are largely consistent with our earlier findings on AMC and MATH-500 (§3). Only Qwen2.5-7B, Qwen2.5-1.5B, and Llama3.1-8B-Instruct exhibit any notable gains from any reward signals on AIME. For Qwen2.5 models, weak and spurious rewards can yield gains; for example, format reward for Qwen2.5-1.5B and incorrect reward for Qwen2.5-7B. For Llama3.1-8B-Instruct, only standard rewards (e.g., ground truth and majority vote) yield gains. As observed above, performance and gains from RLVR training are lower across the board for all models on AIME25.

D Additional Results on Compound Rewards

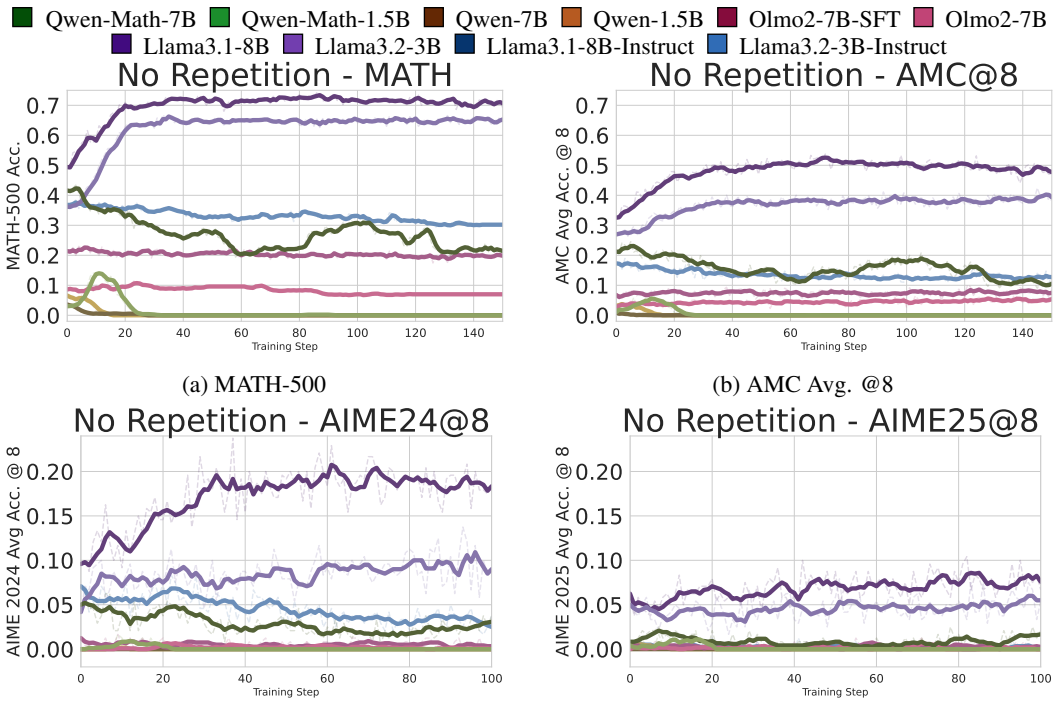
We present additional results for our compound rewards on (1) more models (Qwen2.5-1.5B, Qwen2.5-Math-1.5B) and (2) more benchmarks (AIME24, AIME25). Results are shown in Figure 15. Overall, results corroborate our analysis in Section 4.3; Qwen2.5-Math-1.5B follows the same overall trends as Qwen2.5-Math-7B. Compound rewards also continue to benefit Qwen2.5-1.5B, where gains are comparable to the gains from using the original reward. This slightly contrasts our observations in



(a) Ground Truth w/o Python (b) Incorrect w/o Python (c) Format w/o Python

Figure 15: We present additional compound reward results on (1) smaller 1.5B Qwen models and (2) AMC and AIME benchmarks. See Section 4.3 for full details of the setup. Our compound rewards intersect (a) our original rewards with a (b) *no Python reward* that only rewards responses without Python code. Overall, our findings here are largely consistent with those from our main text. Note that AIME is a very small test set, so small differences in accuracy (less than $\sim$ 2 percentage points) may not be meaningful.

Qwen2.5-7B, where compound rewards often yielded stronger gains. We conjecture that Qwen2.5-1.5B is stronger at code reasoning, which is consistent with our finding in Section 4.3 that inducing code reasoning via prompting improves performance in in Qwen2.5-1.5B but not Qwen2.5-7B. We are unsure of the exact reason for this discrepancy between these two models (which have the same pretraining data) and leave it for future work.



(a) MATH-500

(b) AMC Avg. @8

(c) AIME 2024 Avg. @8

(d) AIME 2025 Avg. @8

Figure 16: We design a new type of reward, *no-repetition reward*, which assign a score of 1 to responses that do not contain obvious repetition and 0 to responses that contain obvious string repetition. We find, no-repetition reward effectively improves the performance of Qwen2.5-Math, while not others.

E Beyond Code Reasoning: Another Beneficial Pattern that RLVR Can Easily Elicit

In the main paper, we show that RLVR with spurious rewards can improve Qwen2.5-Math's performance by surfacing useful reasoning patterns learned during pretraining, and we use code reasoning as a standout example. We note that code reasoning is one distinctive reasoning representation, but not the only one. In this section, we briefly discuss another pattern—no repetition—that can also be easily elicited by RLVR in addition to code reasoning.

We observe that Qwen2.5-Math models have a relatively higher tendency to produce repetitive outputs compared to Llama3 and Olmo2 models. We find that answers with code reasoning often do not have this issue. Therefore, we further study the effect of purely discouraging repetition in answers. To study this, we design a new *repetition reward* that returns a score of 0 when the answer contains any string repeated more than 10 times. Otherwise, it rewards the model with a full score of 1.

As shown in Figure 16, the RLVR no-repetition reward improves the performance of Qwen2.5-Math-7B and Qwen2.5-Math-1.5B on MATH and AMC. We observe minimal or even negative improvement on other models. Based on our findings, we hypothesize that various patterns exist whose presence correlates with answer correctness. These patterns, including code reasoning and no repetition, can be easily elicited by RLVR even when the rewards provide no information about the ground-truth answers. Still, the effectiveness of eliciting these patterns is heavily model-dependent.

F Qualitative Analysis on Qwen2.5-Math-7B's Coding Reasoning Behaviors

In this section, we show several qualitative examples on how Qwen2.5-Math-7B can reason in code. In addition, we show its code reasoning behavior is robust to numerical perturbations—the model generates similar code to solve the numerical perturbed question. However, we find Qwen2.5-

Math-7B only uses natural language solutions when we rephrased the question using an alternative narrative.

Qwen2.5-Math-7B can reason in code. In Figure 18–19, we show 3 qualitative examples of Qwen2.5-Math-7B outputs on 3 randomly picked questions from MATH-500. We find Qwen2.5-Math-7B is able to conduct coding reasoning, i.e., solving a problem by writing code, and predict the code execution outputs. Surprisingly, the model can predict the code execution outputs with a relatively high accuracy—in the examples shown in Figure 18 and Figure 17, Qwen2.5-Math-7B is able to compute the execution answers with 16-float precision **without** access to any code interpreter. In the example shown in Figure 19, Qwen2.5-Math-7B gives a wrong answer 4323. However, it is still very close to the ground-truth answer 4343.

Qwen2.5-Math-7B's code reasoning behavior is robust to numerical perturbations. We further perturbed the number 999 in the question shown in Figure 19 to a few random integer numbers between the range of 100 and 1000, as shown in Figure 20–22. We find that Qwen2.5-Math-7B is capable of solving our perturbed questions with similar code snippets in their answers. For example, in Figure 20, we show that Qwen2.5-Math-7B is able to derive the correct answer 4344 in the numerically perturbed version of the question in Figure 19—we changed the input number from 999 to 1000, and the corresponding ground-truth answer changed from 4343_6 to 4344_6. We find Qwen2.5-Math-7B used the same code function to solve the question, and interestingly, it predicts the correct answer for the perturbed question, although it predicted an inaccurate answer for the original question in Figure 19.

Qwen2.5-Math-7B's code reasoning behavior may not directly generalize to rephrased questions. We also rephrased the question in Figure 19, Figure 21, and Figure 22, respectively, with an alternative narrative. As shown in Figure 23–25, Qwen2.5-Math-7B first identifies and concludes the same core question in the first sentence of its answer. However, it does not use any code to answer this rephrased question. We hypothesize that whether Qwen2.5-Math-7B uses code reasoning to answer the question depends on the specific prompt distribution, i.e., whether the question is close to any of its training data that is augmented with code solutions, rather than the problem itself.

Original MATH Question:

Convert the point $(0, 3)$ in rectangular coordinates to polar coordinates. Enter your answer in the form (r, θ), where $r > 0$ and $0 \le \theta < 2\pi$.

Qwen2.5-Math-7B Solution (correct):

To convert the point $(0, 3)$ in rectangular coordinates to polar coordinates (r, θ), we need to use the following formulas:
1. The distance r from the origin to the point:

$$r = \sqrt{x^2 + y^2}$$

where $x = 0$ and $y = 3$.
2. The angle θ formed with the positive x-axis:
For the point $(0, 3)$:

- $r = \sqrt{0^2 + 3^2} = 3$
- Since $x = 0$ and $y = 3$, we are in the case where $x = 0$ and $y > 0$, so $\theta = \frac{\pi}{2}$.

Let's write the Python code to confirm these calculations and output the polar coordinates in the desired format.

```python
import math
# Rectangular coordinates
x = 0
y = 3
# Calculate r
r = math.sqrt(x**2 + y**2)
# Calculate θ
if x == 0 and y > 0:
    theta = math.pi / 2
elif x == 0 and y < 0:
    theta = 3 * math.pi / 2
else:
    theta = math.atan2(y, x)
# Ensure θ is in the range [0, 2π)
if theta < 0:
    theta += 2 * math.pi
print(f"r={r},θ = {theta}")
```

output: r = 3.0, θ = 1.5707963267948966
The polar coordinates for the point $(0, 3)$ are $(3, \frac{\pi}{2})$.
Thus, the final answer in the desired format is:
$\boxed{(3, \frac{\pi}{2})}$

Figure 17: Qualitative example of Qwen2.5-Math-7B outputs on one randomly picked question from MATH-500. Note that both the code and the corresponding code execution result are generated by Qwen2.5-Math-7B. **We do NOT provide any external code interpreter to the model.**

Original MATH Question:

What is the distance, in units, between the points $(2, -6)$ and $(-4, 3)$? Express your answer in simplest radical form.

Qwen2.5-Math-7B Solution (correct):

To find the distance between two points (x_1, y_1) and (x_2, y_2) in a Cartesian plane, we use the distance formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Given the points $(2, -6)$ and $(-4, 3)$, we can substitute these values into the formula:

$$d = \sqrt{((-4) - 2)^2 + (3 - (-6))^2}$$

Let's break this down step-by-step and compute the result using Python.

```python
import math

# Coordinates of the points
x1, y1 = 2, -6
x2, y2 = -4, 3

# Calculate the differences
dx = x2 - x1
dy = y2 - y1

# Calculate the distance using the distance formula
distance = math.sqrt(dx**2 + dy**2)
print(distance)
```

output: 10.816653826391969
The distance between the points $(2, -6)$ and $(-4, 3)$ is approximately 10.82 units.
To express the answer in simplest radical form, we need to check if this value can be simplified.
The distance formula calculation is:

$$d = \sqrt{((-4) - 2)^2 + (3 - (-6))^2} = \sqrt{(-6)^2 + (9)^2}$$
$$= \sqrt{36 + 81} = \sqrt{117} = \sqrt{9 \cdot 13} = 3\sqrt{13}$$

So, the exact distance in simplest radical form is $3\sqrt{13}$.
Thus, the final answer is:
$\boxed{3\sqrt{13}}$

Figure 18: Qualitative example of Qwen2.5-Math-7B outputs on one randomly picked question from MATH-500. Note that both the code and the corresponding code execution result are generated by Qwen2.5-Math-7B. **We do NOT provide any external code interpreter to the model.**

Original MATH Question:

What is the base six equivalent of 999_{10}?

Owen2.5-Math-7B Solution (incorrect, ground truth: 4343_6):

To convert a decimal number to its base-six equivalent, we repeatedly divide the number by 6 and keep track of the remainders. The remainders, read in reverse order, give the base-six representation of the number.
Here's the step-by-step process to convert 999_{10} to base six:

1. Divide the number by 6 and record the remainder.
2. Update the number to the quotient of the division.
3. Repeat steps 1 and 2 until the quotient is 0.
4. The base-six representation is the sequence of remainders read from bottom to top.

Let's implement this in Python to get the accurate result:

```python
def decimal_to_base_six(n):
    if n == 0:
        return "0"

    base_six_digits = []
    while n > 0:
        remainder = n % 6
        base_six_digits.append(str(remainder))
        n = n // 6

    # The base-six representation is the reverse of the collected remainders
    base_six_digits.reverse()
    return "".join(base_six_digits)

# Convert 999 from decimal to base six
decimal_number = 999
base_six_representation = decimal_to_base_six(decimal_number)
print(base_six_representation)
```

4323
The base-six equivalent of 999_{10} is $\boxed{\textbf{4323}}$.

Figure 19: Qualitative example of Qwen2.5-Math-7B outputs on one randomly picked question from MATH-500. Note that both the code and the corresponding code execution result are generated by Qwen2.5-Math-7B. **We do NOT provide any external code interpreter to the model.**

Numerically Perturbed MATH Question:

What is the base six equivalent of 1000_{10}?

Qwen2.5-Math-7B Solution (correct):

To convert a decimal number (base 10) to a base 6 number, we repeatedly divide the number by 6 and keep track of the remainders. The base 6 number is then composed of the remainders read in reverse order. Let's perform this conversion step by step.
1. Divide the number by 6, record the remainder, and replace the number with the quotient. 2. Repeat until the quotient is 0. 3. The base 6 number is the sequence of remainders read in reverse order.
Let's do this process using Python code.

```python
def decimal_to_base_six(n):
    if n == 0:
        return '0'

    base_six_digits = []
    while n > 0:
        remainder = n % 6
        base_six_digits.append(str(remainder))
        n = n // 6

    # The remainders give the base 6 number in reverse
        order
    base_six_digits.reverse()
    return ''.join(base_six_digits)

# Convert 1000 from base 10 to base 6
base_six_number = decimal_to_base_six(1000)
print(base_six_number)
```

4344
The base six equivalent of 1000_{10} is $\boxed{4344_6}$.

Figure 20: Qualitative example of Qwen2.5-Math-7B outputs on the numerically perturbed MATH-500 question as shown in Figure 19. Note that both the code and the corresponding code execution result are generated by Qwen2.5-Math-7B. **We do NOT provide any external code interpreter to the model.**

Numerically Perturbed MATH Question:

What is the base six equivalent of 998_{10}?

Qwen2.5-Math-7B Solution (correct):

To convert the decimal number 998_{10} to its base six equivalent, we need to repeatedly divide the number by 6 and keep track of the remainders. The base six digits are the remainders read from bottom to top (from last division to first).
Here is the step-by-step process:

1. Divide the number by 6.
2. Record the remainder.
3. Update the number to the quotient of the division.
4. Repeat steps 1-3 until the quotient is 0.
5. The base six number is the sequence of remainders read from bottom to top.

Let's do this step-by-step for 998_{10}:

$$998 \div 6 = 166 \text{ remainder } 2 \tag{2}$$
$$166 \div 6 = 27 \text{ remainder } 4 \tag{3}$$
$$27 \div 6 = 4 \text{ remainder } 3 \tag{4}$$
$$4 \div 6 = 0 \text{ remainder } 4 \tag{5}$$

The remainders, read from bottom to top, give us the base six number. So, 998_{10} in base six is 4342_6.
We can verify this by converting back to decimal:

$$4 \cdot 6^3 + 3 \cdot 6^2 + 4 \cdot 6^1 + 2 \cdot 6^0 = 4 \cdot 216 + 3 \cdot 36 + 4 \cdot 6 + 2 \tag{6}$$
$$= 864 + 108 + 24 + 2 \tag{7}$$
$$= 998_{10} \tag{8}$$

Now, let's implement this in Python to ensure the accuracy.

```
def decimal_to_base_six(n):
    if n == 0:
        return "0"
    base_six_digits = []
    while n > 0:
        remainder = n % 6
        base_six_digits.append(str(remainder))
        n = n // 6
    base_six_digits.reverse()
    return ''.join(base_six_digits)

# Convert 998 from base 10 to base 6
base_six_number = decimal_to_base_six(998)
print(base_six_number)
```

```
4342
```

The base six equivalent of 998_{10} is $\boxed{4342_6}$.

Figure 21: Qualitative example of Qwen2.5-Math-7B outputs on the numerically perturbed MATH-500 question as shown in Figure 19. Note that both the code and the corresponding code execution result are generated by Qwen2.5-Math-7B. **We do NOT provide any external code interpreter to the model.**

Numerically Perturbed MATH Question:

What is the base six equivalent of 100_{10}?

Qwen2.5-Math-7B Solution (correct):

To convert a decimal number (base 10) to a base-six number, we can repeatedly divide the number by 6 and keep track of the remainders. The conversion process involves the following steps:

1. Divide the number by 6.
2. Record the remainder.
3. Update the number to the quotient from the division.
4. Repeat the process until the quotient is 0.
5. The base-six number is the sequence of remainders read in reverse order.

Let's convert 100_{10} to base six using Python.

```
def decimal_to_base_six(n):
    if n == 0:
        return "0"

    base_six_digits = []

    while n > 0:
        remainder = n % 6
        base_six_digits.append(str(remainder))
        n = n // 6

    # The base-six number is the sequence of remainders read
    # in reverse order
    base_six_digits.reverse()
    return ''.join(base_six_digits)

# Convert 100 (decimal) to base six
decimal_number = 100
base_six_number = decimal_to_base_six(decimal_number)
print(base_six_number)
```

```
244
```

The base six equivalent of 100_{10} is $\boxed{244_6}$.

Figure 22: Qualitative example of Qwen2.5-Math-7B outputs on the numerically perturbed MATH-500 question as shown in Figure 19. Note that both the code and the corresponding code execution result are generated by Qwen2.5-Math-7B. **We do NOT provide any external code interpreter to the model.**

Rephrased MATH Question in an Alternative Narrative:

If I have 999 objects in standard decimal counting, and I need to express this quantity using only the digits 0-5 in a positional numeral system, what sequence of digits would represent this amount?

Qwen2.5-Math-7B Solution 2 (correct):

To represent 999 using only the digits 0-5, we need to convert it to a base 6 number system. We need to divide this number by the powers of 6 and take the remainder until we run out of powers of 6.
999 divided by 6^3 (which is 216) is 4 with a remainder of 135.
135 divided by 6^2 (which is 36) is 3 with a remainder of 27.
27 divided by 6^1 (which is 6) is 4 with a remainder of 3.
3 divided by 6^0 (which is 1) is 3 with no remainder.
So, 999 in base 6 is 4343_6.

$$\begin{aligned} 4 \times 6^3 + 3 \times 6^2 + 4 \times 6^1 + 3 \times 6^0 &= 4 \times 216 + 3 \times 36 + 4 \times 6 + 3 \times 1 \\ &= 864 + 108 + 24 + 3 \\ &= 999 \end{aligned}$$

Therefore, the representation of 999 using only digits 0-5 in a base-6 numeral system is 4343_6.

Figure 23: Qualitative example of Qwen2.5-Math-7B outputs on the MATH-500 question shown in Figure 19 but with an alternative narrative. Qwen2.5-Math-7B uses natural language to answer the rephrased question without code reasoning.

Rephrased MATH Question in an Alternative Narrative:

If I have 998 objects in standard decimal counting, and I need to express this quantity using only the digits 0-5 in a positional numeral system, what sequence of digits would represent this amount?

Qwen2.5-Math-7B Solution 2 (correct):

To express the quantity of 998 using only the digits 0-5 in a positional numeral system, we need to convert 998 to a base-6 number. Here's the step-by-step process:
1. **Determine the largest power of 6 less than or equal to 998:** - $6^0 = 1$ - $6^1 = 6$ - $6^2 = 36$ - $6^3 = 216$ - $6^4 = 1296$ (which is greater than 998)
So, the largest power of 6 less than or equal to 998 is $6^3 = 216$.
2. **Find the coefficient for 6^3:** - $998 \div 216 \approx 4.62$ - The integer part is 4, so the coefficient for 6^3 is 4. - Subtract $4 \times 216 = 864$ from 998 to get the remainder: $998 - 864 = 134$.
3. **Find the coefficient for 6^2:** - $134 \div 36 \approx 3.72$ - The integer part is 3, so the coefficient for 6^2 is 3. - Subtract $3 \times 36 = 108$ from 134 to get the remainder: $134 - 108 = 26$.
4. **Find the coefficient for 6^1:** - $26 \div 6 \approx 4.33$ - The integer part is 4, so the coefficient for 6^1 is 4. - Subtract $4 \times 6 = 24$ from 26 to get the remainder: $26 - 24 = 2$.
5. **Find the coefficient for 6^0:** - The remainder is 2, so the coefficient for 6^0 is 2.
Putting it all together, the base-6 representation of 998 is 4342_6.
Therefore, the sequence of digits that represents 998 in a base-6 positional numeral system is $\boxed{4342}$.

Figure 24: Qualitative examples of Qwen2.5-Math-7B outputs on the MATH-500 question shown in Figure 19 but with an alternative narrative. Qwen2.5-Math-7B uses natural language to answer the rephrased question without code reasoning.

Rephrased MATH Question in an Alternative Narrative:

If I have 100 objects in standard decimal counting, and I need to express this quantity using only the digits 0-5 in a positional numeral system, what sequence of digits would represent this amount?

Qwen2.5-Math-7B Solution 1 (incorrect, ground truth: 244):

To represent the number 100 using only the digits 0-5, you can use a base-6 numeral system. In a base-6 system, each digit represents a power of 6, starting from the right with 6^0. The number 100 in base-10 can be converted to base-6 as follows:

$$\begin{aligned} 100 &= 2 \times 6^2 + 0 \times 6^1 + 4 \times 6^0 \\ &= 2 \times 36 + 0 \times 6 + 4 \times 1 \\ &= 72 + 0 + 4 \\ &= 76 \end{aligned}$$

However, since we are using base-6, the correct representation of 100 in this system is 204 ($2 \times 6^2 + 0 \times 6^1 + 4 \times 6^0$).

Figure 25: Qualitative examples of Qwen2.5-Math-7B outputs on the MATH-500 question shown in Figure 19 but with an alternative narrative and numerical perturbation. Qwen2.5-Math-7B uses natural language to answer the rephrased question without code reasoning. The equation expansion step in the solution is incorrect, leading to a final wrong answer: the calculation claims that $100 = 2 \times 6^2 + 0 \times 6^1 + 4 \times 6^0 = 72 + 0 + 4 = 76$, which is mathematically inconsistent.