PREDICTING REWARD HACKING IN RLVR REQUIRES ACCOUNTING FOR GRADIENT GEOMETRY

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement Learning from Verifiable Rewards (RLVR) has driven rapid LLM progress in math and coding, where verifying solutions is cheap. To move beyond these domains, RLVR is now commonly used with *imperfect verifiers* such as LLM judges. Yet training with imperfect verifiers can degrade final task performance as models learn to reward hack, i.e., to exploit verifier errors rather than fully solve the task. It is therefore desirable to predict the impact of verifier errors on task performance before committing substantial compute. Prior work suggests using confusion matrix statistics like Youden's index. However, we prove that when verifier errors are systematic, these statistics are insufficient even for simple log-linear policies. Motivated by this limitation, we advocate for using information about the policy's gradient geometry to improve predictions. We provide initial evidence about the promise of this direction through a simple gradient-based prediction procedure: we approximate the base policy's tangent kernel using sampled generations and reuse this kernel to efficiently simulate linearized RLVR dynamics across many verifiers. All simulated policies can then be assessed on the same set of ground-truth labels, so benchmarking further verifiers is cheap. For models up to 8B parameters, we show that this linearized simulation already improves task performance prediction in many settings.

1 INTRODUCTION

The last two years have seen unprecedented progress in the ability of large language models (LLMs) to write code and solve well-defined mathematical problems. Coding assistants went from curiosity to ubiquity, and OpenAI claims that an internal model solved the Navier–Stokes Millennium Prize Problem (OpenAI, 2026). Much of this progress has been driven by reinforcement learning from verifiable rewards (RLVR), where models are trained to optimize a typically binary reward provided by a *verifier* such as a suite of unit tests, a proof checker or a comparison to known ground truth (Guo et al., 2025; Zeng et al., 2025; Lambert et al., 2024).

However, verifiers that perfectly capture downstream task performance are often unavailable or expensive to query: Faulty code can pass unit tests, checking mathematical proofs requires expert judgment or a faithful formalization in Lean, and obtaining ground truth at scale is challenging for domains like experimental science. Instead, RLVR is commonly used with cheap but highly *imperfect* verifiers based on heuristics such as token overlap with a reference answer (Zheng et al., 2025), LLM judges (Ma et al., 2025) or rubrics (Gunjal et al., 2025).

This approach is risky as verifier errors open up the possibility of *reward hacking*: Instead of learning to perform the downstream task, a model trained to optimize against an imperfect verifier might learn to exploit verifier errors. Practitioners thus face a prediction problem: Rather than wasting resources on training runs that fail due to reward hacking, they would like to detect reward hacking *before* committing large amounts of compute. Prior work has suggested solving this using metrics based on the verifier's initial confusion matrix, such as Youden's index (Rad et al., 2026) or precision (Dorner et al., 2026). However, we show that confusion matrices are insufficient to predict downstream performance when errors are systematic. Specifically, two verifiers with *the same* confusion matrix can lead to complete task failure and strong task performance, respectively (illustrated in Fig. 1a).

We argue that this failure is due to the confusion matrix hiding important information: which particular completions are mislabeled and how parameter updates transfer between completions. This

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

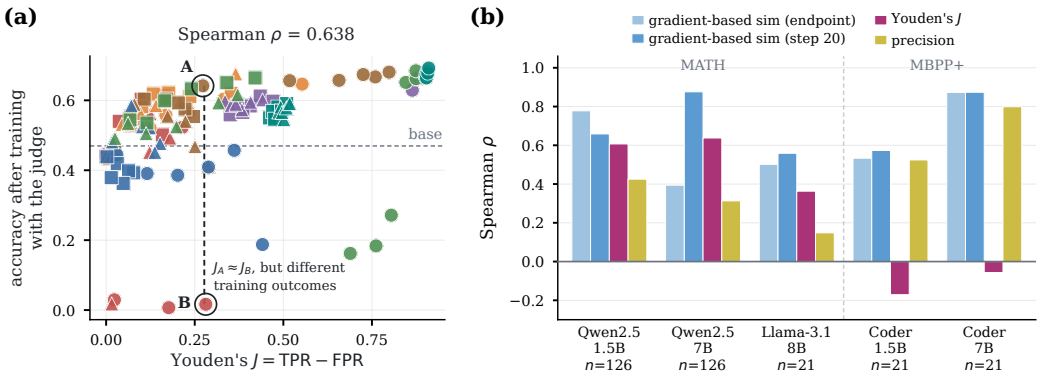


Figure 1: **(a)** Youden's index $J = \mathrm{TPR} - \mathrm{FPR}$ of each of 126 LLM judges, against the real accuracy that Qwen2.5-7B reaches on MATH after RLVR with that judge. Colors and shapes mark the judge model and template (legend in Fig. 3), and the dashed line marks the base model's accuracy. **(b)** Rank correlation between predictions and post-RL ground truth accuracy (n verifiers per setting). The gradient-based simulation outperforms baselines in most cases, especially when stopped early.

is encoded in the geometry of the policy's gradients: Two completions with similar gradients are jointly reinforced, while reinforcing one completion makes completions with opposite gradients less likely. Intuitively, accounting for these effects requires information about the policy's gradients.

We practically demonstrate the utility of gradient information using a simple method: we first generate a fixed *bank* of completions with the base model. We then approximate the base model by a log-linear policy on the bank, and use inner products between bank sample gradients to define a frozen tangent kernel, which allows us to estimate how RLVR redistributes probability mass within the bank. Ground-truth scores for the bank samples allow us to evaluate the simulated policies, providing predictions of real post-RL performance. Crucially, for a fixed task and base model, the bank, kernel and ground-truth scores can be reused across a large set of verifiers.

We evaluate this approximation using 315 RLVR runs across five model-task pairs, with model sizes ranging from 1.5B to 8B parameters and various LLM judges. In four out of five settings, our simulation outperforms all evaluated confusion matrix baselines in terms of rank correlation with post-RLVR performance (Fig. 1b). These results highlight the promise of using gradient geometry to predict reward hacking and motivate further work on more robust prediction methods. In summary, our contributions are as follows:

1. For two verifiers with the same initial confusion matrix, we prove that the exact policy gradient method with the same fully expressive log-linear parameterization and initialization yields oracle accuracies of 1 and 0, respectively (Proposition 1). Thus, any metric based on the confusion matrix alone is in principle insufficient to predict reward hacking.
2. We propose a simple kernel method that simulates RLVR trajectories to predict post-RL performance and can efficiently reuse completions and gradient computations across verifiers (Section 5). For 315 RLVR runs across five model-task pairs with LLM judge verifiers, these predictions outperform confusion-matrix statistics in most settings (Fig. 1b).

2 Related Work

Reinforcement learning from verifiable rewards using algorithms such as GRPO (Shao et al., 2024) has become a common approach for improving LLM reasoning capabilities. It is used both in domains with access to strong verifiers (Shao et al., 2024; Guo et al., 2025; Zeng et al., 2025; Lambert et al., 2024) and beyond exactly verifiable domains, relying on LLM judges or rubrics as reward signals (Ma et al., 2025; Gunjal et al., 2025).

The impact of verifier errors Reinforcement learning with an imperfect reward signal can cause reward hacking as models learn to exploit reward errors rather than improving on the actual task

(Skalse et al., 2022; Gao et al., 2023). That said, recent works report substantial robustness of RLVR to synthetically injected random verifier errors (Lv et al., 2025; Plesner et al., 2026). Rad et al. (2026) support this observation theoretically, showing that for random verifier errors and a restricted policy class, a positive value of Youden's index $J = \mathrm{TPR} - \mathrm{FPR}$ preserves task learning.

However, when verifier errors are systematic, this result need not hold. Indeed, Egashira et al. (2026) empirically show that systematic false positives can cause training to plateau or collapse depending on specific error patterns, not just error rates. Similarly, Zhu & Kang (2026) find that incorrect annotations meaningfully degrade RLVR performance, while Huang et al. (2026) find that LLM judge verifiers with strong static accuracy are still vulnerable to reward hacking.

Evaluating reward functions for RLHF Prior work has considered benchmarking reward models that are used for RLHF post-training. While Lambert et al. (2025) focused on how well rewards rank actions, others also investigated how well their benchmark scores correlate with post-RLHF performance (Liu et al., 2025a; Frick et al., 2025; Kim et al., 2025). Interestingly, both Malik et al. (2026) and Wen et al. (2025) find that while ranking accuracy is predictive of downstream Best-of-N performance using the reward model, its predictive power for PPO performance is much weaker.

Razin et al. (2025) provide a partial explanation: As rewards in RLHF are not binary, the magnitude of the difference between two actions' rewards matters. Further, Shang et al. (2026) prove that outputs with intermediate rewards can slow down convergence of log-linear policies. Note that these explanations cannot apply to RLVR - studied in this paper - that uses binary verifiers such that magnitude is fixed and intermediate rewards do not exist.

Gradient geometry for understanding training dynamics A parameter update based on one completion can affect other completions, depending how the completions' gradients align. The neural tangent kernel (NTK) captures these interactions to first order (Jacot et al., 2018; Lee et al., 2020). While it has historically been used to study *supervised* LLM fine-tuning (Malladi et al., 2023), the NTK has also recently been applied to explain increased model confidence during reinforcement learning (Tomihari, 2026). Relatedly, Wang et al. (2026b) observe approximately linear parameter trajectories during RLVR, and propose linear extrapolation to improve training efficiency.

Beyond that, gradient geometry has been used for reward shaping and mitigating reward hacking. Ackermann et al. (2026) use gradient regularization to steer optimization towards flatter optima associated with better downstream performance, while Wang et al. (2026a) cluster gradients to detect reward hacking. Most closely related, Zheng et al. (2026) use the NTK to motivate their method, giving a bonus reward to answers that are both accepted by the verifier and align with expert solutions in terms of gradient direction. In contrast, our work focuses on using gradient geometry to predict RLVR performance *before* training, rather than directly mitigating reward hacking during training.

3 Problem Setup

In this section, we introduce our model of RLVR with imperfect verifiers and the resulting prediction problem. Following Rad et al. (2026), we consider rewards assigned to full model responses, such that prompts are contexts and model responses are actions in a *contextual bandit problem*. Formally, for a finite prompt set Q, sampled uniformly during training, we model the LLM as a policy $\pi_\theta(\cdot \mid q)$ over completions $c \in \mathcal{A} = \Sigma^{\leq L}$, with token vocabulary Σ, length cap L, and parameters $\theta \in \mathbb{R}^d$.

RLVR optimizes the expected value of a binary *verifier reward* $r : \mathcal{A} \times Q \to \{0, 1\}$, which is provided by a verifier such as a test suite, a deterministic extraction rule, or an LLM judge. Our analysis focuses on the training dynamics of the plain policy-gradient update (Williams, 1992)

$$\theta_{t+1} = \theta_t + \eta \nabla_\theta \mathbb{E}_{q \sim \mathrm{Unif}(Q), c \sim \pi_{\theta_t}(\cdot|q)}[r(c, q)], \tag{1}$$

which matches, up to constants, the expected update of a slightly simplified Dr. GRPO algorithm (Liu et al., 2025b), a debiased version of classical GRPO (Shao et al., 2024). Downstream task performance is determined by the *oracle reward* $r^\star : \mathcal{A} \times Q \to \{0, 1\}$.

We write $\pi_t := \pi_{\theta_t}$ and abbreviate the expectation of any function $f : \mathcal{A} \times Q \to \mathbb{R}$ of the completion c and the prompt q at training step t as

$$\mathbb{E}_t[f] := \mathbb{E}_{q \sim \mathrm{Unif}(Q), c \sim \pi_t(\cdot|q)}[f(c, q)], \qquad \mathbb{E}_t[f \mid q] := \mathbb{E}_{c \sim \pi_t(\cdot|q)}[f(c, q)].$$

Similarly, when taking gradients, we always read $\mathbb{E}_t[f]$ and $\mathbb{E}_t[f \mid q]$ as functions of the policy parameters θ_t. Then we can write shorthands for relevant quantities such as the expected reward, prompt-conditional expected reward, and expected task performance at time t as follows

$$\underbrace{V_t := \mathbb{E}_t[r]}_{\text{verifier acceptance rate}}, \qquad \underbrace{V_t(q) := \mathbb{E}_t[r \mid q]}_{\text{conditional acceptance rate}}, \qquad \underbrace{U_t := \mathbb{E}_t[r^\star]}_{\text{accuracy}}. \tag{2}$$

Informally, reward hacking means that U_t stagnates or even decreases, while V_t increases. Our goal is thus to predict the RLVR performance U_T after T steps using a given verifier without actually running RLVR. This would help to identify verifiers that are likely to cause reward hacking, as well as screen candidate verifiers for downstream training.

A natural starting point for this is to characterize verifier errors: the reward and oracle disagree on false positives ($r = 1$, $r^\star = 0$) and false negatives ($r = 0$, $r^\star = 1$). At time t, they then induce the *policy-weighted confusion matrix* $C_t(r) = (\mathbb{E}_t[\mathbf{1}\{r = i, r^\star = j\}])_{i,j\in\{0,1\}}$, which is commonly summarized by the true-positive rate $\mathrm{TPR}_t = \frac{\mathbb{E}_t[rr^\star]}{\mathbb{E}_t[r^\star]}$, the false-positive rate $\mathrm{FPR}_t = \frac{\mathbb{E}_t[r(1-r^\star)]}{\mathbb{E}_t[1-r^\star]}$, as well as Youden's index $J_t = \mathrm{TPR}_t - \mathrm{FPR}_t$. The latter has indeed been shown to determine the success of RLVR training for a restricted policy class under random verifier errors (Rad et al., 2026).

However, note that intuitively, an RL update usually shifts probability mass from rejected completions to accepted ones (also see Eq. (3) later on), such that the confusion matrix $C_t(r)$ is not generally constant in time. Consequently, as we prove in the next section, *no statistic* based on the initial policy's confusion matrix C_0 is, in general, sufficient to predict U_T. Correspondingly, the confusion matrix is also insufficient to screen a set of verifiers for reward hacking.

4 Limits of confusion-matrix statistics

In this section we prove that even for log-linear policies, predicting performance purely based on confusion matrices is impossible. The proof sketch for the failure mode then also provides intuition on how gradient information can crucially help to prevent it.

4.1 Provable Insufficiency for log-linear policies

To provide a clean exposition, we prove insufficiency on the class of log-linear policies $\pi_\theta(\cdot \mid q) = \mathrm{softmax}(\Phi_q\theta)$ where $\Phi_q \in \mathbb{R}^{|\mathcal{A}|\times d}$ are fixed query-features. We call a log-linear policy class π_θ *fully expressive* if, for every collection of full-support distributions $\{\mu(\cdot \mid q)\}_{q\in Q}$, there exists θ such that $\pi_\theta(\cdot \mid q) = \mu(\cdot \mid q)$, for all $q \in Q$. The following proposition shows that starting with the same initial policy, policy gradient iterations using two different verifiers with identical confusion matrices can lead to perfect accuracy for one verifier, and zero accuracy for the other.

Proposition 1. *For every finite nonempty prompt set Q and completion set $\mathcal{A}$ with $|\mathcal{A}| \geq 4$, there exists a fully expressive log-linear policy class, an oracle $r^\star$, and two verifiers $r^{(1)}, r^{(2)}$ such that, for every $u, j \in (1/2, 1)$, there is a common initialization θ_0 such that*

$$U_0 = u, \qquad C_0(r^{(1)}) = C_0(r^{(2)}), \qquad J_0(r^{(1)}) = J_0(r^{(2)}) = j.$$

For $k \in \{1, 2\}$, let $\pi_t^{(k)}, V_t^{(k)}, U_t^{(k)}$ denote the trajectory of the quantities defined in Eq. (2) obtained by running exact policy gradient for an arbitrary fixed step size $\eta > 0$ (Eq. (1)) on $r^{(k)}$. Then,

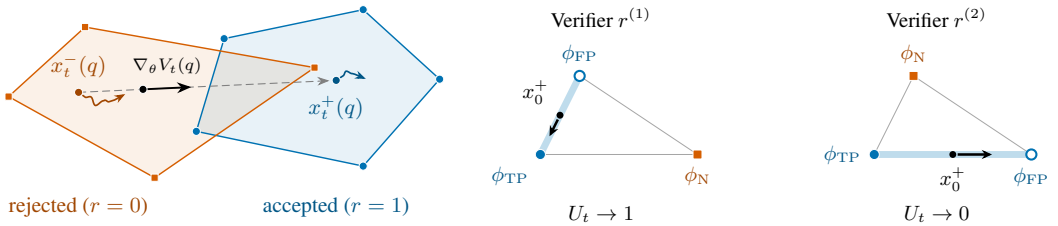
$$U^{(1)} \to 1, \qquad U^{(2)} \to 0.$$

The full proof can be found in Appendix A. We now aim to provide intuition for this failure mode by constructing an example in a simple visualizable setting. Before doing that, we would like to first point out a few mathematical facts about log-linear policies that simplify our subsequent explanation.

First, to prove convergence of the policy gradient updates to points with certain accuracy, a key quantity to control is the direction of the policy update. Concretely, for $0 < V_t(q) < 1$ we can write

$$\Delta\theta_t = \eta\mathbb{E}_q[V_t(q)(1 - V_t(q))(x_t^+(q) - x_t^-(q))], \tag{3}$$

where $x_t^+(q) := \frac{\mathbb{E}_t[r\, x_t|q]}{V_t(q)}$, $x_t^-(q) := \frac{\mathbb{E}_t[(1-r)\, x_t|q]}{1-V_t(q)}$ and for a completion c and a prompt q, $x_t(c, q) := \nabla_\theta \log \pi_t(c \mid q)$ is the *gradient feature vector*. Thus, $x_t^+(q)$ and $x_t^-(q)$ are the mean gradient features



(a) Geometry of policy gradient update

(b) Same confusion matrix, opposite outcomes.

Figure 2: **(a)** Circles/squares represent gradient features of accepted/rejected completions, respectively, and $x_t^+(q)/x_t^-(q)$ are their policy-weighted barycenters. The black arrow shows $\nabla_\theta V(\theta_t, q)$ (Eq. (3)). Curved arrows illustrate how policy reweighting moves barycenters within their respective convex hulls. **(b)** Let $\phi_a = (0,0)$, $\phi_b = (1,2)$, $\phi_c = (4,0)$, with $\theta_0 = 0$ and $r^\star = 1$ only for a. Verifiers $r^{(1)}$, $r^{(2)}$ accept $\{a,b\}$ and $\{a,c\}$, respectively, and have identical initial confusion matrices. We draw x_t^+ in fixed ϕ-coordinates, omitting the common shift $-\mathbb{E}_t[\phi]$, which does not affect differences. Arrows show the self-reinforcing drift of x_0^+ toward the TP under $r^{(1)}$ and the FP under $r^{(2)}$.

of the accepted and rejected completions for query q, each weighted by π_t within the respective set. Geometrically, $x_t^\pm(q) \in \mathbb{R}^d$ is a point in the convex hull spanned by all accepted or rejected gradient features respectively (Fig. 2a). The parameter update in Eq. (3) is hence in the direction of a weighted average of $x_t^+ - x_t^-$ over queries.

For a general policy, the direction $x_t^+(q) - x_t^-(q)$ changes both as the policy reweighs completions (where the mean gradient feature of each subset lies within the convex hull) and as the gradient features $x_t(c,q)$ themselves (the extreme points of the convex hull) move. For log-linear policies with fixed query-features Φ_q, the gradient features stay constant relative to one another, such that we only need to track the relative locations of $x_t^\pm(q)$ within the convex hulls. Concretely, let $\phi(c,q)$ be the c-th row of Φ_q. Then we have $x_t(c,q) = \phi(c,q) - \mathbb{E}_t[\phi \mid q]$ and therefore

$$x_t^+(q) - x_t^-(q) = \frac{\mathbb{E}_t[r\ \phi \mid q]}{V_t(q)} - \frac{\mathbb{E}_t[(1-r)\ \phi \mid q]}{1 - V_t(q)}.$$

4.2 Proof intuition and helpfulness of gradients

We provide a simplified example based on ideas from the full proof of Proposition 1 in Appendix A. Consider a fixed prompt, $|\mathcal{A}| = 3$ possible completions and the log-linear policy $\pi_\theta = \text{softmax}(\Phi\theta)$. To see how verifiers with identical confusion matrices can cause different optimization outcomes, we will compare the trajectories of two verifiers that both output one true positive TP, one false positive FP, and a rejected completion N. Consider the trajectory of one such verifier and denote the log-odds by $\ell_t := \log \frac{\pi_t(\text{TP})}{\pi_t(\text{FP})}$. If $\Delta\ell_t > 0$, then the true positive share $\frac{\pi_t(\text{TP})}{\pi_t(\text{TP})+\pi_t(\text{FP})}$ increases, moving x_t^+ closer to ϕ_{TP}. Now, suppose that at some point $\Delta\theta_t \propto x_t^+ - x_t^-$ has positive inner product with $\phi_{\text{TP}} - \phi_{\text{FP}}$, as in the left panel of Fig. 2b. Then $\Delta\ell_t > 0$, and the resulting movement of x_t^+ makes this alignment even more positive. This self-reinforcing effect drives $\ell_t \to \infty$. Since $V_t \to 1$ in the construction, it follows that $\pi_t(\text{TP}) \to 1$ and hence $U_t \to 1$.[1] A negative alignment produces the symmetric feedback toward FP, yielding $U_t \to 0$.

Proposition 1 emphasizes the fundamental limitation of confusion-matrix statistics. They only record how much probability mass lies in each error class. The proof intuition illustrates how verifiers with the same error proportions can still lead to vastly different policy updates and trajectories and final accuracies. First-order information additionally captures interactions between the labels and gradient updates, allowing us to understand how reinforcing a completion affects other completions. For log-linear policies, gradient information would indeed be sufficient to fully trace the trajectory and mitigate such ambiguities. This insight motivates us to investigate whether gradient information can also help to predict performance of RLVR for complex policy classes such as

[1] The full proof shows that $\sum_t \eta V_t(1 - V_t) = \infty$, such that the log-odd increments diverge.

large LLMs. For this purpose, in the next section we propose a simple prediction method based on log-linear policy approximation that can be used for arbitrary policy classes.

5 TANGENT-KERNEL SIMULATION

Confusion-matrix statistics discard how verifier labels interact with gradient geometry. Earlier we noted that the parameter update direction $x_t^+ - x_t^-$ can change for two reasons: first, the policy reweights completions and second, the gradient features $x_t(c, q)$ themselves change. For log-linear policies only the first matters. We use this as an approximation, fixing gradient features of the base policy π_0 and approximating policy gradient updates assuming a log-linear model p_θ. This yields a simulator whose geometry is computed once and can be reused across verifiers.

Completion bank and log-linear model. For each prompt q, we independently sample M completions $c_{q,1}, \ldots, c_{q,M} \sim \pi_0(\cdot \mid q)$ forming a *bank* of completions c. We then compute the base gradient features $x_0(c_{q,m}, q)$ for all completions $c_{q,m}$ in the bank. Let $X_q \in \mathbb{R}^{M \times d}$ be the matrix whose m-th row is $x_0(c_{q,m}, q)^\top$. We approximate the policy π_θ by the log-linear model

$$p_\theta(\cdot \mid q) := \mathrm{softmax}(X_q(\theta - \theta_0)), \qquad \theta \in \mathbb{R}^d. \tag{4}$$

Therefore, the support of the policy p_θ is limited to completions $c_{q,m}$ from the bank. Because the bank is sampled from π_0, uniform averages over its elements provide Monte Carlo estimates of base-policy expectations. In particular, $p_0 := p_{\theta_0}$ is uniform over the bank and therefore approximates the base model acceptance rate. Additionally, p_θ preserves the base policy's first-order log-odds geometry on the bank. Namely, for any two sampled completions $c_{q,m}$ and $c_{q,n}$ of the same prompt,

$$\nabla_\theta \log \frac{p_0(c_{q,m} \mid q)}{p_0(c_{q,n} \mid q)} = x_{q,m} - x_{q,n} = \nabla_\theta \log \frac{\pi_0(c_{q,m} \mid q)}{\pi_0(c_{q,n} \mid q)}. \tag{5}$$

Kernel dynamics. Fix a verifier r and write $r_{q,m} := r(c_{q,m}, q)$ and let $X := [X_q]_{q \in Q}$. We now run the exact policy-gradient update on the empirical log-linear policy:

$$\theta_{t+1} = \theta_t + \frac{\eta_{\text{sim}}}{|Q|} X^\top g_t,$$

where $g_t(q, m) := p_t(c_{q,m} \mid q)(r_{q,m} - \widehat{V}_t(q))$ and $\widehat{V}_t(q) := \sum_{m=1}^M p_t(c_{q,m} \mid q) r_{q,m}$ is the mass on accepted completions for prompt q at time t. Moving to the codomain of X, we can set $w_0 = 0$ and

$$w_{t+1} = w_t + \frac{\eta_{\text{sim}}}{|Q|} g_t, \tag{6}$$

and obtain $\theta_t - \theta_0 = X^\top w_t$. Therefore $p_t(\cdot \mid q) = \mathrm{softmax}([XX^\top w_t]_q)$. The gram matrix $K = XX^\top$ is the empirical tangent kernel of the base LLM on the bank. Its cross-prompt entries describe how an update induced by one prompt changes the logits of completions for another prompt. To reduce memory use, we apply CountSketch to the gradients (Charikar et al., 2002) before constructing X. Further details are given in Appendix F.5.

Verifier prediction. For a verifier r, we run the simulation as in Eq. (6). We use the simulated policy at time T, $p_T^{(r)}$ to approximate the distribution of the trained model on the bank. Note that this only requires the gram matrix K and the verifier labels on the bank. We can then use ground-truth labels $r^\star_{q,m}$ to obtain the *predicted accuracy* as

$$\widehat{U}_T^{(r)} := \sum_{q \in Q} \sum_{m=1}^M p_T^{(r)}(c_{q,m} \mid q) r^\star_{q,m} / |Q|. \tag{7}$$

This can be further approximated by importance sampling with a subsample of labels $r^\star_{q,m}$ when ground truth is particularly expensive. Given a pool of verifiers $\{r_1, \ldots, r_R\}$, we can then use $\widehat{U}_T^{(r)}$ to obtain a ranking without paying for R real training runs.

Scope of the approximation. The empirical log-linear model departs from full RLVR training in two ways. First, Eq. (5) only holds exactly at θ_0, whereas the gradient features and the gram matrix K drift in real training. Second, p_t only redistributes its mass on the finite bank that was sampled from the base model, and cannot represent policies that leave the bank's support.

6 Experiments on LLMs

In this section, we experimentally demonstrate that accounting for the policy's gradient geometry can improve predictions of post-RLVR performance, compared to using confusion-matrix statistics. To assess how well different predictors rank candidate verifiers, we focus on the Spearman correlation between the predictions $\widehat{U}_T^{(r)}$ and post-RLVR accuracy, across a fixed set of verifiers.

6.1 Experimental details

Models, tasks, and verifiers. We focus on LLM judges as practically important examples of verifiers with imperfect rewards and assess them on five model-task pairs: Qwen2.5-1.5B, Qwen2.5-7B and Llama-3.1-8B on the MATH dataset (Hendrycks et al., 2021), and Qwen2.5-Coder-1.5B and Qwen2.5-Coder-7B on MBPP (Austin et al., 2021). In general, we are interested in the setting where ground truth verifiers are hard to come by. However for valid evaluation of our predictions, we rely on datasets like these where we in fact have explicit oracle verifiers: symbolic answer equivalence via `math_verify`, and the MBPP+ unit tests (Liu et al., 2023) (see Appendix B for details).

We use 1024 prompts on MATH and 377 prompts on MBPP for both training and bank construction. To construct the bank, we sample 512 completions/prompt for Qwen2.5-7B/MATH and Qwen2.5-Coder-7B/MBPP+, and 256 completions/prompt for the remaining three model-task pairs (see Appendix F.1 for ablations on bank size). We use LLM judge verifiers prompted to directly produce a binary yes/no verdict. They accept whenever the normalized probability of the judge to return the YES token, $P(\text{YES})/[P(\text{YES}) + P(\text{NO})]$, exceeds a threshold τ. Overall, our experiments involve 7 judge models (0.5B-12B) with 3 prompt templates each, and a range of 6 acceptance thresholds τ. We run RLVR for the full pool of 126 verifiers on MATH for Qwen2.5-7B and Qwen2.5-1.5B and a randomly sampled threshold for each model-prompt combination giving 21 judges on the remaining three model-task pairs. More details on the judge models and prompts are in Appendix C.

Training. We use Dr. GRPO (Liu et al., 2025b), a debiased variant of group relative policy optimization (GRPO) (Shao et al., 2024). For a prompt q, let $c_1, \dots, c_G \sim \pi_\theta(\cdot \mid q)$ be a sampled group with rewards $r_i := r(c_i, q)$ and group mean $\bar{r}$. We omit importance sampling and KL penalization and instead use the simplified update rule

$$\theta_{t+1}^{\text{Dr.}} = \theta_t^{\text{Dr.}} + \frac{\eta}{G} \sum_{i=1}^{G} \frac{(r_i - \bar{r})}{L} \nabla_\theta \log \pi_\theta(c_i \mid q), \tag{8}$$

Up to a factor of $\frac{1}{L}(1 - \frac{1}{G})$, this is an unbiased estimator of the policy-gradient step in Eq. (1). We account for this correction factor in the simulation by setting $\eta_{\text{sim}} = \frac{\eta(G-1)}{LG}$. Following Mukherjee et al. (2026) we use SGD, and fix the learning rate to $\eta = 0.1$. For more details, see Appendix D.

Evaluation. Using the oracle verifiers on MATH/MBPP, we can evaluate post-RLVR accuracy. On MATH, we estimate accuracy using 8 completions on a fixed subset of 128 training prompts. The same prompts are used across all training runs, keeping prompt difficulty constant across runs. Note that these prompts are in-sample to isolate *generalization* as a separate problem. On MBPP the outcome is measured by grading all rollouts from the final training step (24 prompts $\times$ 8 completions). We compute the confusion-matrix statistics on the same bank used for simulation.

Ranking LLM-judge verifiers We evaluate the predicted accuracy $\widehat{U}_T$ against final accuracy after RLVR (Fig. 1b). Our $\widehat{U}_T$ attains mean rank correlation of 0.616, compared with 0.276 for Youden's J, and 0.442 for precision, respectively, and exceeds both on four of five pairs. Qwen2.5-7B on MATH is the exception, with simulation/post-RL rank correlation dropping to 0.394 (Fig. 4 second plot). For the remaining settings, Fig. 3 shows Youden's J and our prediction $\widehat{U}_T$ against post-RL accuracy.

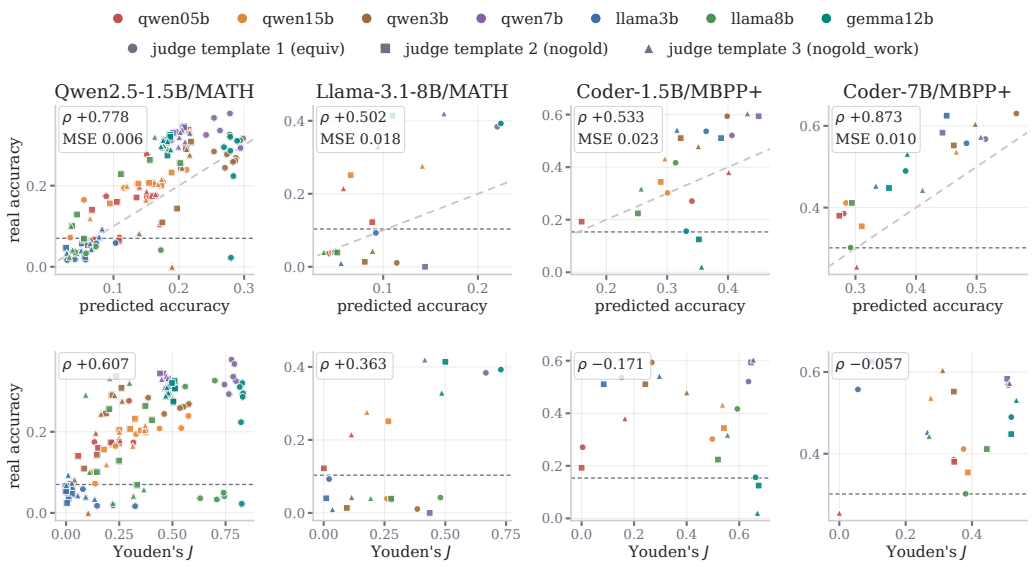


Figure 3: Predicted accuracy $\widehat{U}_T$ versus real oracle accuracy on four model-task pairs. Each point is one verifier. The diagonal dashed line marks equality, and the horizontal line marks the base model's accuracy U_0. Results for Qwen2.5-7B/MATH are shown in Fig. 4.

Early stopping. As our simulation produces a full trajectory of performance estimates $\widehat{U}_t^r$ rather than just an estimate $\widehat{U}_T^r$ for the final time step T, we can in principle use earlier estimates to predict final RLVR performance. Perhaps surprisingly, this indeed produces stronger results: Reading out the simulation results $\widehat{U}_{20}^r$ at step 20 yields an average rank correlation of 0.708, while using each verifier's peak $\max_t \widehat{U}_t^{(r)}$ yields 0.692, both substantially above the 0.616 achieved by the full simulation. These gains are largely driven by a stark improvement for Qwen2.5-7B/MATH, the only setting in which the endpoint $\widehat{U}_T$ underperformed the confusion-matrix baselines. This is consistent with the frozen kernel using a local approximation around θ_0, such that errors can accumulate along the simulation trajectory. Another potential explanation for the stronger performance of early readouts is that later simulated policies $p_t(\cdot \mid q)$ often become dominated by a small number of completions in the bank. For example, for Qwen2.5-7B on MATH, the verifier-median perplexity drops from 226 at step 20 to 3 at step 300. Consider Appendix F.2 for a more detailed investigation of the readout step's impact on the simulation's diversity and prediction quality.

7 Discussion

Finally, we conclude with a critical discussion of our results that give rise to avenues for future work.

7.1 Limitations of the simulation and future work

Even though we demonstrate that our simple frozen-kernel simulation can improve predictions upon confusion-matrix statistics, the Spearman correlations are far from perfect. We investigate two sources of error whose mitigation provides promising directions towards better simulation accuracy in future work: The finite sample bank, and drift in the gradient geometry during RLVR.

Finite bank. As the completion bank used by our simulation is drawn from the base policy, it may fail to cover completions that are initially rare but become more likely during RLVR. To investigate the impact of coverage on simulation quality, we run the following experiment for Qwen2.5-7B/MATH: For each verifier, we construct a *patched* bank. Concretely, for each prompt, we sample a completion for the post-RLVR policy corresponding to the verifier and use it to replace a random completion in the original bank with the same ground-truth label. Gradients for the new completions

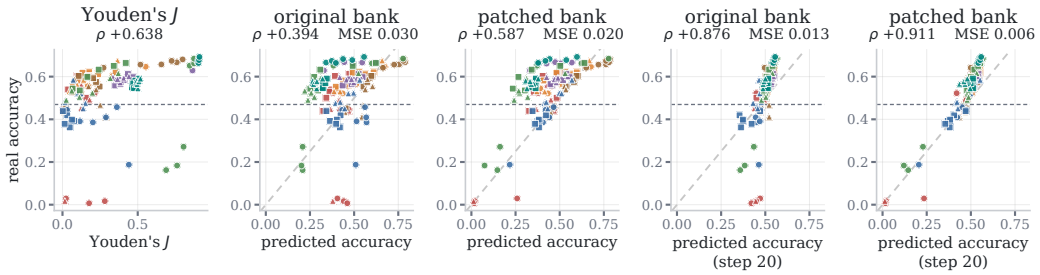


Figure 4: Original versus patched bank on Qwen2.5-7B/MATH. Each point is one of the 126 verifiers, plotted against its final accuracy U_T. Panels, left to right: Youden's index; the predicted accuracy $\widehat{U}_T$ on the original and on the patched bank; $\widehat{U}_{20}$ on the original and on the patched bank. Each panel reports the Spearman correlation ρ with U_T and, for the simulations, the mean squared error (MSE) against U_T. The patched panels are a post-hoc diagnostic, not predictions.

are then recomputed at the initial policy's parameters. Patching raises the Spearman correlation from 0.394 to 0.587 for the full simulation and from 0.876 to 0.911 for the early stopped simulation at step 20. Interestingly, this improvement is one-sided: As shown in Fig. 4, patching greatly reduces instances of reward hacking missed by the simulation, while doing little in terms of reducing false alarms. See Appendix F.3 for additional detail.

A tempting way to improve coverage is to sample a larger bank from the base policy. However, a separately drawn bank with $M = 4096$ completions per prompt slightly raises ρ at step 20 relative to its eight disjoint $M = 512$ subsets (0.880 versus 0.860 on average) and lowers it for the full simulation (0.249 versus 0.374, Appendix F.1). For the verifiers the simulation already under-predicts, the larger bank lowers $\widehat{U}_T$ further.

Gradient geometry drift. The simulation operates with a fixed kernel calculated at θ_0, and is thus not able to account for changes in gradient alignment during training. To investigate the impact of this inherent limitation of fixed-feature approximation, we run another post-hoc experiment for seven handpicked Qwen2.5-7B/MATH verifiers that represent simulation failures and healthy runs. We recompute the kernel using the post-RLVR parameters θ_{300} on a subset of the initial bank's completions, restricted to the 128 evaluation prompts, and rerun the simulation with this kernel. While the original kernel based on θ_0 for the same prompts yields simulation results that are almost uncorrelated with post-RLVR performance, the recomputed kernel achieves near-perfect ranking ($\rho = 0.964$). Interestingly, a simulation readout at $t = 20$ produces the exact same, slightly worse, ranking for both kernels ($\rho = 0.929$). These results are consistent with initial gradient geometry being informative for short, but not longer simulation trajectories. For more details, see Appendix F.4.

Together, these diagnostics point to incomplete coverage of emerging false-positives and changing gradient geometry as key limitations of the long-horizon simulation. Future work could address these, for example by regularizing RLVR, reducing drift in both coverage and gradient geometry.

7.2 Conclusion

Our theoretical results clearly establish that predicting reward hacking in RLVR requires information beyond the verifier's initial confusion matrix. Empirically, our basic frozen-kernel simulation performs well, outperforming confusion matrix baselines in most settings. However, it lacks accuracy for Qwen2.5-7B on MATH, and it remains unclear whether insufficient bank coverage, shifting gradient geometry, or a combination of both is to blame. Beyond that, our experiments do not cover models larger than 8B parameters or RLVR with more sophisticated optimizers than SGD.

Taken together, these points indicate that while the presented gradient geometric approach shows great promise, there is ample potential for future work to improve, extend, and robustify it.

AI USE STATEMENT

In this work, we used generative AI tools to formulate mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, and to support qualitative and thematic data analysis. We have not used generative AI tools to generate synthetic data sets, help to develop theoretical models or conceptual frameworks, assist with translation, clean or reformat datasets, interpret results, or to provide critical ingredients for proving mathematical claims. Additionally, we have used generative AI tools to create scientific figures or images, suggest experimental parameters, create or edit software code, give feedback on the writing, identify, summarize or analyze relevant existing literature, brainstorm, source/search for information, format references, suggest a structure for a research paper. We have reviewed all AI-assisted work. All references and literature summaries were verified against the original sources, AI-assisted code was reviewed and tested for correctness by the authors, suggested experimental parameters and feedback on experiments were evaluated by the authors and validated empirically, figures were checked against the underlying results, and all AI-drafted or AI-edited text was critically reviewed and revised by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We document experimental details, hyperparameters, training frameworks and seeds in detail. Code will be published at a later point.

REFERENCES

Johannes Ackermann, Michael Noukhovitch, Takashi Ishida, and Masashi Sugiyama. Gradient Regularization Mitigates Reward Hacking in Reinforcement Learning from Human Feedback and Verifiable Rewards. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732*, 2021.

Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding Frequent Items in Data Streams. In *Automata, Languages and Programming*, Lecture Notes in Computer Science, pp. 693–703. Springer, 2002.

Ethan Chern, Haoyang Zou, Xuefeng Li, Jiewen Hu, Kehua Feng, Junlong Li, and Pengfei Liu. Generative AI for Math: Abel. `https://github.com/GAIR-NLP/abel`, 2023.

Florian E. Dorner, Yatong Chen, André F. Cruz, and Fanny Yang. ROC-n-reroll: How verifier imperfection affects test-time scaling. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2026.

Kazuki Egashira, Mark Vero, Jasper Dekoninck, Florian E. Dorner, Robin Staab, and Martin Vechev. Delay, Plateau, or Collapse: Evaluating the Impact of Systematic Verification Error on RLVR. In *Proceedings of the Conference on Language Modeling (COLM)*, 2026.

Evan Frick, Tianle Li, Connor Chen, Wei-Lin Chiang, Anastasios Angelopoulos, Jiantao Jiao, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. How to Evaluate Reward Models for RLHF. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

Leo Gao, John Schulman, and Jacob Hilton. Scaling Laws for Reward Model Overoptimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as Rewards: Reinforcement Learning Beyond Verifiable Domains. *arXiv preprint arXiv:2507.17746*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, (8081):633–638, 2025. ISSN 1476-4687.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

Yuzhen Huang, Weihao Zeng, Xingshan Zeng, Qi Zhu, and Junxian He. From Accuracy to Robustness: A Study of Rule- and Model-based Verifiers in Mathematical Reasoning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural Tangent Kernel: Convergence and Generalization in Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8571–8580, 2018.

Sunghwan Kim, Dongjin Kang, Taeyoon Kwon, Hyungjoo Chae, Dongha Lee, and Jinyoung Yeo. Rethinking Reward Model Evaluation Through the Lens of Reward Overoptimization. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 13252–13280, 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. *arXiv preprint arXiv:2411.15124*, 2024.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. RewardBench: Evaluating Reward Models for Language Modeling. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 1755–1797, 2025.

Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent. *Journal of Statistical Mechanics: Theory and Experiment*, (12):124002, December 2020. ISSN 1742-5468. arXiv:1902.06720 [stat.ML].

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 21558–21572, 2023.

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. RM-Bench: Benchmarking Reward Models of Language Models with Subtlety and Style. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025a.

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. In *Proceedings of the Conference on Language Modeling (COLM)*, 2025b.

Ang Lv, Ruobing Xie, Xingwu Sun, Zhanhui Kang, and Rui Yan. The Climb Carves Wisdom Deeper Than the Summit: On the Noisy Rewards in Learning to Reason. *arXiv preprint arXiv:2505.22653*, 2025.

Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhu Chen. General-Reasoner: Advancing LLM Reasoning Across All Domains. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Morrison, Noah Smith, Hanna Hajishirzi, and Nathan Lambert. RewardBench 2: Advancing Reward Model Evaluation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2026.

Sadhika Malladi, Alexander Wettig, Dingli Yu, Danqi Chen, and Sanjeev Arora. A Kernel-Based View of Language Model Fine-Tuning. In *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of Machine Learning Research, pp. 23610–23641. PMLR, 2023.

Sagnik Mukherjee, Lifan Yuan, Pavan Jayasinha, Dilek Hakkani-Tür, and Hao Peng. Do We Need Adam? Surprisingly Strong and Sparse Reinforcement Learning with SGD in LLMs. *arXiv preprint arXiv:2602.07729*, 2026.

OpenAI. On the Navier–Stokes Millennium Prize Problem. Technical report, OpenAI, September 2026. URL https://cdn.openai.com/pdf/32d9f210-8b73-45e0-91bc-82a30aef8a9a/navier-stokes.pdf. Accessed: 2026-09-23.

Andreas Plesner, Francisco Guzmán, and Anish Athalye. An Imperfect Verifier is Good Enough: Learning with Noisy Rewards. *arXiv preprint arXiv:2604.07666*, 2026.

Ali Rad, Khashayar Filom, Darioush Keivan, Peyman Mohajerin Esfahani, and Ehsan Kamalinejad. Rate or Fate? RLV$^{\varepsilon}$R: Reinforcement Learning with Verifiable Noisy Rewards. *arXiv preprint arXiv:2601.04411*, 2026.

Noam Razin, Zixuan Wang, Hubert Strauss, Stanley Wei, Jason D Lee, and Sanjeev Arora. What Makes a Reward Model a Good Teacher? An Optimization Perspective. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Shuning Shang, Hubert Strauss, Stanley Wei, Sanjeev Arora, and Noam Razin. When Errors Can Be Beneficial: A Categorization of Imperfect Rewards for Policy Gradient. *arXiv preprint arXiv:2604.25872*, 2026.

Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, Yulia Tsvetkov, Hannaneh Hajishirzi, Pang Wei Koh, and Luke Zettlemoyer. Spurious Rewards: Rethinking Training Signals in RLVR. *arXiv preprint arXiv:2506.10947*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300*, 2024.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and Characterizing Reward Hacking. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 9460–9471, 2022.

Akiyoshi Tomihari. Learning Dynamics in RL Post-Training for Language Models. *arXiv preprint arXiv:2601.04670*, 2026.

Songtao Wang, Quang Hieu Pham, Fangcong Yin, Xinpeng Wang, Jocelyn Qiaochu Chen, Greg Durrett, and Xi Ye. Detecting and Suppressing Reward Hacking with Gradient Fingerprints. In *Proceedings of the Conference on Language Modeling (COLM)*, 2026a.

Tianle Wang, Jiayu Liu, Zhongyuan Wu, Shenghao Jin, Wei Chen, Hao Xu, and Ning Miao. Linear Dynamics in the RLVR Training of Large Language Models. *arXiv preprint arXiv:2601.04537*, 2026b.

Xueru Wen, Jie Lou, Yaojie Lu, Hongyu Lin, Xing Yu, Xinyu Lu, Ben He, Xianpei Han, Debing Zhang, and Le Sun. Rethinking Reward Model Evaluation: Are We Barking up the Wrong Tree? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3–4):229–256, 1992. ISSN 1573-0565.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-Math Technical Report: Toward Mathematical Expert Model via Self-Improvement. *arXiv preprint arXiv:2409.12122*, 2024.

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. In *Proceedings of the Conference on Language Modeling (COLM)*, 2025.

Leqi Zheng, Jinbo Su, Fang Niu, Chaokun Wang, Weiping Wang, Jiajun Zhang, Shannan Yan, Jie Wu, Zhaolu Kang, Rong Fu, and Hang Zhang. Gradients Know What Outcomes Don't: Unlocking Reinforcement Learning for LLM Reasoning with Gradient-Aligned Rewards. *arXiv preprint arXiv:2609.03342*, 2026.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. DeepResearcher: Scaling Deep Research via Reinforcement Learning in Real-world Environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 414–431, 2025.

Yuxuan Zhu and Daniel Kang. Noisy Data is Destructive to Reinforcement Learning with Verifiable Rewards. *arXiv preprint arXiv:2603.16140*, 2026.

A Full proof of Proposition 1

Proof of Proposition 1. We first prove the result for one prompt and four completions using the same construction as in the sketch, and then extend the construction to arbitrary finite Q and $|\mathcal{A}| \geq 4$.

Four-action construction. Let $\mathcal{A} = \{a, b, c, d\}$ and define $r^\star = \mathbf{1}_{\{a\}}$, $r^{(b)} = \mathbf{1}_{\{a,b\}}$, and $r^{(c)} = \mathbf{1}_{\{a,c\}}$. Thus a is the unique oracle-correct completion, while b and c are the respective false positives of the two verifiers. For $p, j \in (1/2, 1)$, set $\alpha := (1-p)(1-j), \beta := (1-p)(2j-1)$. Both quantities are positive and $p + 2\alpha + \beta = 1$. We initialize the policy at $\pi_0 = (p, \alpha, \alpha, \beta)$. For either verifier,

$$(\mathrm{TP}, \mathrm{FP}, \mathrm{FN}, \mathrm{TN}) = (p, \alpha, 0, \alpha + \beta) = (p, (1-p)(1-j), 0, j(1-p)).$$

Consequently,

$$U_0 = p, \qquad \mathrm{TPR}_0 = 1, \qquad \mathrm{FPR}_0 = \frac{\alpha}{1-p} = 1 - j, \qquad J_0 = j,$$

and the two initial confusion matrices are identical. Consider the log-linear policy

$$\pi_\theta = \mathrm{softmax}(0, 2u + w, u + 2v, u - 2v), \qquad \theta = (u, v, w) \in \mathbb{R}^3,$$

with feature vectors $\phi_a = (0, 0, 0), \phi_b = (2, 0, 1), \phi_c = (1, 2, 0), \phi_d = (1, -2, 0)$. The three nonzero feature vectors are linearly independent. Hence all three log-odds relative to a can be chosen independently, so the policy is fully expressive. To explicitly construct that desired initialization, let

$$L_\alpha := \log \frac{\alpha}{p}, \qquad L_\beta := \log \frac{\beta}{p},$$

and choose

$$u_0 = \frac{L_\alpha + L_\beta}{2}, \qquad v_0 = \frac{L_\alpha - L_\beta}{4}, \qquad w_0 = -L_\beta.$$

Then

$$2u_0 + w_0 = L_\alpha, \qquad u_0 + 2v_0 = L_\alpha, \qquad u_0 - 2v_0 = L_\beta,$$

which gives $\pi_{\theta_0} = (p, \alpha, \alpha, \beta)$.

Divergence of steps. We use the following fact for both verifier trajectories. For a fixed binary verifier, let

$$h_t := \eta V_t (1 - V_t), \qquad H_t := \sum_{s=0}^{t-1} h_s.$$

We claim that $H_t \to \infty$. Indeed, the policy-gradient update (Eq. (1)) can be written as

$$\theta_{t+1} - \theta_t = h_t \big(\phi_t^+ - \phi_t^-\big),$$

where ϕ_t^+ and ϕ_t^- are the conditional means of the accepted and rejected completions repectively. This is possible due to the cancellation of the normlization by $\mathbb{E}_t[\phi]$ inside x_t^+ and x_t^- (from Eq. (3)). Since both belong to the convex hull of the finite set $\{\phi\}_{a,b,c,d}$, there exists $D < \infty$ such that $\|\theta_{t+1} - \theta_t\| \le D h_t$ for every t. If $\sum_t h_t < \infty$, then θ_t would converge to a finite limit θ_∞. A finite softmax parameter assigns positive probability to every completion. Since both the accepted and rejected sets are nonempty, this would imply $0 < V(\theta_\infty) < 1$, and hence

$$h_t \to \eta V(\theta_\infty)\big(1 - V(\theta_\infty)\big) > 0,$$

contradicting $\sum_t h_t < \infty$. This proves $H_t \to \infty$ for every $\eta > 0$.

Training on $r^{(b)}$. We suppress the superscript (b) in this paragraph and define the conditional shares

$$\rho_t := \frac{\pi_t(b)}{\pi_t(a) + \pi_t(b)} \qquad \lambda_t := \frac{\pi_t(c)}{\pi_t(c) + \pi_t(d)}.$$

The accepted and rejected feature means are $\phi_t^+ = \rho_t \phi_b$ and $\phi_t^- = \lambda_t \phi_c + (1 - \lambda_t)\phi_d$. Therefore,

$$\begin{pmatrix} \Delta u_t \\ \Delta v_t \\ \Delta w_t \end{pmatrix} = h_t \begin{pmatrix} 2\rho_t - 1 \\ 2 - 4\lambda_t \\ \rho_t \end{pmatrix}. \tag{9}$$

Let $\ell_t := \log \frac{\pi_t(b)}{\pi_t(a)} = 2u_t + w_t$ be the log-odds between b and a. Note that $\rho_t = \sigma(\ell_t)$, where σ is the logistic function. Now Eq. (9) gives $\Delta\ell_t = h_t(4\rho_t - 2 + \rho_t) = h_t(5\rho_t - 2)$. At initialization, $\rho_0 = \frac{\alpha}{p+\alpha} < \frac{1}{3}$, because, $j > 1/2$ and $p > 1/2$ imply $\alpha < (1-p)/2 < p/2$ and hence $2\alpha < p$. If $\rho_t \le \rho_0$, then

$$\Delta\ell_t \le h_t\left(\frac{5}{3} - 2\right) = -\frac{1}{3} h_t < 0.$$

Because $\rho_t = \sigma(\ell_t)$ is increasing in ℓ_t, induction therefore gives $\rho_{t+1} < \rho_t \le \rho_0 < \frac{1}{3}$ for every t. Summing the preceding bound and using the divergence of H_t yields

$$\ell_t \le \ell_0 - \frac{1}{3} H_t \to -\infty.$$

Thus $\pi_t(b)/\pi_t(a) \to 0$. The first coordinate in Eq. (9) similarly satisfies

$$\Delta u_t = h_t(2\rho_t - 1) \le -\frac{1}{3} h_t,$$

so

$$u_t \longrightarrow -\infty. \tag{10}$$

It remains to control the relative probabilities of the two rejected completions. Let $\gamma_t := \log \frac{\pi_t(c)}{\pi_t(d)} = 4v_t$, so that $\lambda_t = \sigma(\gamma_t)$. Then Eq. (9) gives

$$\Delta\gamma_t = 8h_t(1 - 2\lambda_t) = -8h_t \tanh(\gamma_t/2).$$

The increment always points toward zero, and $h_t \le \eta/4$ implies $|\Delta\gamma_t| \le 2\eta$. If an update does not cross zero, it decreases $|\gamma_t|$; if it crosses zero, the resulting magnitude is at most 2η. Hence $|\gamma_t| \le \max\{|\gamma_0|, 2\eta\}$ for every t, so v_t remains bounded. Combining this with Eq. (10),

$$\log \frac{\pi_t(c)}{\pi_t(a)} = u_t + 2v_t \to -\infty, \qquad \log \frac{\pi_t(d)}{\pi_t(a)} = u_t - 2v_t \to -\infty.$$

Together with $\ell_t \to -\infty$, this gives $\pi_t^{(b)}(a) \to 1$. Since a is oracle-correct and accepted by $r^{(b)}$, it follows that $(V_t^{(b)}, U_t^{(b)}) \to (1, 1)$.

Training on $r^{(c)}$. We now suppress the superscript (c) and define

$$\rho_t := \frac{\pi_t(c)}{\pi_t(a) + \pi_t(c)} \qquad \zeta_t := \frac{\pi_t(b)}{\pi_t(b) + \pi_t(d)}.$$

The accepted and rejected feature means are $\phi_t^+ = \rho_t \phi_c$ and $\phi_t^- = \zeta_t \phi_b + (1-\zeta_t)\phi_d$. Consequently,

$$\begin{pmatrix} \Delta u_t \\ \Delta v_t \\ \Delta w_t \end{pmatrix} = h_t \begin{pmatrix} \rho_t - 1 - \zeta_t \\ 2(1 + \rho_t - \zeta_t) \\ -\zeta_t \end{pmatrix}. \tag{11}$$

Let $\ell_t := \log \frac{\pi_t(c)}{\pi_t(a)} = u_t + 2v_t$ be the log-odds between c and a, so that $\rho_t = \sigma(\ell_t)$. Then Eq. (11) gives

$$\Delta \ell_t = h_t(3 + 5\rho_t - 5\zeta_t), \tag{12}$$

$$\Delta w_t = -h_t \zeta_t \leq 0,$$

$$\Delta(\ell_t - 5w_t) = h_t(3 + 5\rho_t) \geq 3h_t. \tag{13}$$

By the divergence of H_t, $\ell_t - 5w_t \to \infty$. Because w_t is nonincreasing, there are two cases. If w_t is bounded below, it converges to a finite limit, and Eq. (13) immediately implies $\ell_t \to \infty$. Otherwise, $w_t \to -\infty$. To handle this case, observe that

$$\frac{\zeta_t/(1-\zeta_t)}{\rho_t/(1-\rho_t)} = \frac{\pi_t(b)/\pi_t(d)}{\pi_t(c)/\pi_t(a)} = e^{w_t}. \tag{14}$$

Once $w_t \leq 0$, Eq. (14) implies $\zeta_t \leq \rho_t$, and Eq. (12) then gives $\Delta \ell_t \geq 3h_t$. Since every tail of $\sum_t h_t$ diverges, we again obtain $\ell_t \to \infty$. Thus $\pi_t(c)/\pi_t(a) \to \infty$ in both cases. It remains to show that c also dominates the two rejected completions. From Eq. (11),

$$\Delta \log \frac{\pi_t(c)}{\pi_t(b)} = h_t(5 + 3\rho_t - 2\zeta_t) \geq 3h_t, \tag{15}$$

$$\Delta \log \frac{\pi_t(c)}{\pi_t(d)} = 8h_t(1 + \rho_t - \zeta_t) \geq 8h_t\rho_t. \tag{16}$$

By Eq. (15) and the divergence of H_t, $\log \frac{\pi_t(c)}{\pi_t(b)} \to \infty$. Moreover, $\ell_t \to \infty$ implies $\rho_t = \sigma(\ell_t) \to 1$. Hence, eventually $\rho_t \geq 1/2$, and Eq. (16) gives $\Delta \log \frac{\pi_t(c)}{\pi_t(d)} \geq 4h_t$, so $\log \frac{\pi_t(c)}{\pi_t(d)} \to \infty$. Completion c consequently dominates every other completion, i.e. $\pi_t^{(c)}(c) \to 1$. Since c is an oracle-incorrect completion accepted by $r^{(c)}$, it follows that $(V_t^{(c)}, U_t^{(c)}) \to (1, 0)$.

Extension to $|\mathcal{A}| > 4$. Let $N := |\mathcal{A}| \geq 4$ and $L := N - 3$, and write $\mathcal{A} = \{a, b, c, d_1, \ldots, d_L\}$. We split the role of d among L symmetric true negatives: keep $r^\star = \mathbf{1}_{\{a\}}$, $r^{(b)} = \mathbf{1}_{\{a,b\}}$, and $r^{(c)} = \mathbf{1}_{\{a,c\}}$ as before, and initialize $\pi_0(a) = p$, $\pi_0(b) = \pi_0(c) = \alpha$, and $\pi_0(d_\ell) = \beta/L$. The same confusion-matrix calculation applies because the total mass of the d_ℓ actions is β.

To obtain a fully expressive parameterization while preserving the four-action dynamics, let $S_L := \{z \in \mathbb{R}^L : \sum_{\ell=1}^L z_\ell = 0\}$ and $\psi_\ell := e_\ell - \frac{1}{L}\mathbf{1} \in S_L$, where for $L = 1$ we interpret $S_L = \{0\}$. Take parameters $(u, v, w, z) \in \mathbb{R}^3 \times S_L$ and features

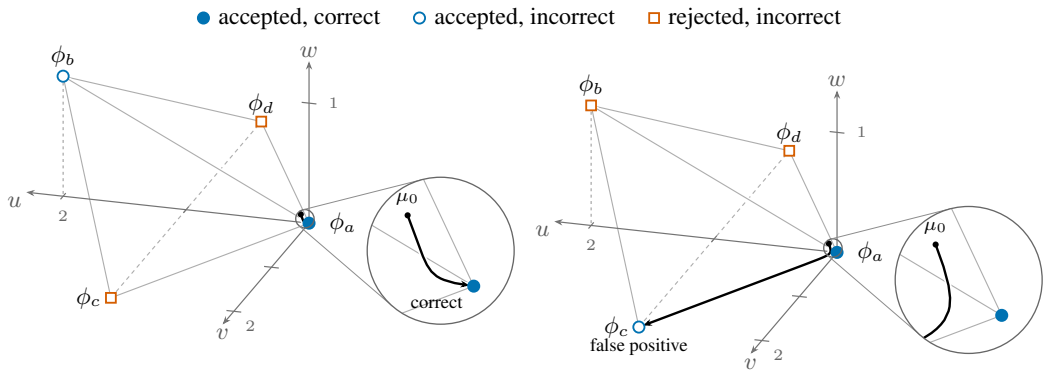
$$\phi_a = (0, 0, 0, 0), \quad \phi_b = (2, 0, 1, 0), \quad \phi_c = (1, 2, 0, 0), \quad \phi_{d_\ell} = (1, -2, 0, \psi_\ell).$$

These features are fully expressive. Indeed, suppose $\lambda_b\phi_b + \lambda_c\phi_c + \sum_{\ell=1}^L \lambda_\ell \phi_{d_\ell} = 0$. The S_L component implies that all λ_ℓ are equal to some common value λ. The w coordinate then gives $\lambda_b = 0$, while the v and u coordinates give $\lambda_c = L\lambda$ and $\lambda_c + L\lambda = 0$, respectively. Thus $\lambda = 0$ and all coefficients vanish, so the $N - 1$ feature differences relative to a are linearly independent.

Set $z_0 = 0$ and choose (u_0, v_0, w_0) to realize the probabilities above. Because the d_ℓ have identical labels and equal probabilities, their mean S_L feature is $\frac{1}{L}\sum_{\ell=1}^L \psi_\ell = 0$. The policy gradient therefore has zero S_L component whenever $z_t = 0$. By induction, $z_t = 0$ for every t, and the d_ℓ remain equiprobable.

Their aggregate probability is proportional to $L\exp(u_t - 2v_t) = \exp(u_t - 2v_t + \log L)$. Consequently, after the constant change of coordinates $\bar{u}_t = u_t + \frac{1}{2}\log L$, $\bar{v}_t = v_t - \frac{1}{4}\log L$, $\bar{w}_t = w_t - \log L$, the aggregate logits of a, b, c, and $\{d_1, \ldots, d_L\}$ are 0, $2\bar{u}_t + \bar{w}_t$, $\bar{u}_t + 2\bar{v}_t$, and $\bar{u}_t - 2\bar{v}_t$. Thus their aggregate dynamics are exactly the four-action dynamics proved above, and it follows that $\pi_t^{(b)}(a) \to 1$ and $\pi_t^{(c)}(c) \to 1$ for every $N \geq 4$.

$$C_0\big(r^{(b)}\big) = C_0\big(r^{(c)}\big), \qquad J_0 = j.$$



(a) oracle success, $r^{(b)}$ accepts $\{a, b\}$

(b) reward hacking, $r^{(c)}$ accepts $\{a, c\}$

Figure 5: Setup of the full proof of Proposition 1. The barycenter $\mu_0 = \sum_k \pi_t(k)\phi(k)$ depicts the trajectory for the two verifiers. For $r^{(1)}$ we obtain $\pi_t(a) \to 1$ and therefore $U^{(b)} \to 1$, while for $r^{(2)}$ induces $\pi_t(c) \to 1$ and therefore $U^{(c)} \to 0$.

Extension to arbitrary prompt sets. Finally, let Q be any nonempty finite prompt set. Give each prompt an independent copy of the preceding feature space: if ϕ_c denotes the single-prompt feature of completion c, define $\phi(c, q) := e_q \otimes \phi_c$. The resulting log-linear policy is fully expressive because the log-odds for every prompt can be chosen independently. Use the same oracle, verifiers, and initial completion probabilities at every prompt.

Since prompts are sampled uniformly, the parameter block associated with each prompt follows the single-prompt dynamics with effective step size $\eta/|Q|$. This is positive and constant, so the preceding arguments apply to every prompt. Averaging over prompts therefore gives

$$(V_t^{(b)}, U_t^{(b)}) \longrightarrow (1, 1), \qquad (V_t^{(c)}, U_t^{(c)}) \longrightarrow (1, 0).$$

This completes the proof. □

B GROUND-TRUTH ORACLES

We elaborate on the implementation details of the oracle reward function for the different datasets.

Mathematics. We score answer equivalence with `math_verify`, as wrapped by verl's `reward_score.math_verify.compute_score`. The gold answer is the content of the *last* `\boxed` expression in the reference solution, and a completion is correct if its extracted answer is verified equivalent to that gold. Bank labels and the oracle reward that is logged during training are produces using the same function.

Program synthesis. We extract a program from the completion's fenced code block following verl's rule in `reward_score.sandbox_fusion`. A completion with no fenced block scores zero without being executed. We then score a program by running the MBPP+ extended testsuite (Liu et al., 2023) with a 30 second limit, and a timeout or any raised exception scores zero.

C LLM JUDGES

Models. Every judge is an instruction-tuned model pinned by revision. In particular, we use the following models (Hugging Face model ids): Qwen/Qwen2.5-0.5B-Instruct, Qwen/Qwen2.5-1.5B-Instruct, meta-llama/Llama-3.2-3B-Instruct, Qwen/Qwen2.5-3B-Instruct, Qwen/Qwen2.5-7B-Instruct, meta-llama/Llama-3.1-8B-Instruct, google/gemma-3-12b-it.

Judge templates. Each judge model is paired with three templates. The placeholders are filled as follows: {question} is the full prompt the policy saw, {candidate} the completion truncated at 24000 characters to fit in the judge's context, and {gold} the reference answer. In the case of program synthesis, we pass the first three base tests each cut at 400 characters as {gold}.

Listing 1: (MATH, equiv)

```
You are grading a solution to a math problem. You are given the prompt
    the student saw, the reference answer, and the student's solution.

The student was given:
{question}

Reference answer:
{gold}

Student's solution:
{candidate}

Grade the answer the solution finally arrives at.

Question: Is that answer mathematically equivalent to the reference
    answer?
Reply with exactly one word: YES or NO.
```

Listing 2: (MATH, nogold)

```
You are grading a solution to a math problem. You are given the prompt
    the student saw and the student's solution. No reference answer is
    available.

The student was given:
{question}

Student's solution:
{candidate}

Grade the answer the solution finally arrives at.

Question: Is that answer correct?
Reply with exactly one word: YES or NO.
```

Listing 3: (MATH, nogold_work)

```
You are grading a solution to a math problem. You are given the prompt
    the student saw and the student's solution. No reference answer is
    available.

The student was given:
{question}

Student's solution:
{candidate}

Grade the answer the solution finally arrives at.

Question: Check the solution's working. Is the answer it reaches correct
    ?
Reply with exactly one word: YES or NO.
```

Note that on MBPP+, the policy's prompt already contains the problem's base tests, so every template shows them to the judge: equiv repeats the first three as {gold}, and the nogold templates' statement that no tests are available is inaccurate.

Listing 4: (MBPP+, `equiv`)

```
You are grading a solution to a programming problem. You are given the
    prompt the
student saw, some of the tests it must pass, and the student's program.

The student was given:
{question}

Some of the tests:
{gold}

Student's program:
{candidate}

Grade the program the student finally provides.

Question: Would that program pass these tests and the problem's
    remaining tests?
Reply with exactly one word: YES or NO.
```

Listing 5: (MBPP+, `nogold`)

```
You are grading a solution to a programming problem. You are given the
    prompt the
student saw and the student's program. No tests are available.

The student was given:
{question}

Student's program:
{candidate}

Grade the program the student finally provides.

Question: Is that program correct?
Reply with exactly one word: YES or NO.
```

Listing 6: (MBPP+, `nogold_work`)

```
You are grading a solution to a programming problem. You are given the
    prompt the
student saw and the student's program. No tests are available.

The student was given:
{question}

Student's program:
{candidate}

Grade the program the student finally provides.

Question: Check the program's logic and its edge cases. Is it correct?
Reply with exactly one word: YES or NO.
```

Verdicts and decisions. A filled template is sent to the judge as a single user message in the judge's own chat template. We run one-token inference at temperature 0, and we read the log-probabilities of its 1024 most likely first tokens. Let A be the total probability of the accept tokens `YES`, `Yes` and `yes`, each with and without a leading space, and R the same for `NO`. The judge accepts a completion if $A/(A+R) > \tau$. Probability on all other tokens is ignored, and a completion with no verdict token among the top 1024 is rejected. For MBPP+, a completion without a code block is rejected without querying the judge, using the same code extraction as the unit-test oracle.

Note however, that the judge still judges the full completion and not only the extracted code. During training the judge is served with vLLM in bfloat16.

As the 6 thresholds we use $(0.1, 0.2, 0.3, 0.5, 0.7, 0.9)$.

D TRAINING

	MATH			MBPP+	
	Qwen2.5-1.5B	Qwen2.5-7B	Llama-3.1-8B	Coder-1.5B	Coder-7B
difficulty levels	3–5	3–5	1–2	–	–
policy prompt	Abel	Qwen	Abel	MBPP	MBPP
bank prompts P	1024	1024	1024	377	377
completions per prompt m	256	512	256	256	512
batch size (prompts)	48	24	120	24	24
training steps	300	300	200	150	150
epochs over the bank	14.1	7.0	23.4	9.5	9.5
validation every (steps)	20	20	13	–†	–†
completion cap L	4096	4096	4096	1024	1024
format gate	–	–	–	yes	yes
shared by all					
objective	Dr. GRPO: advantage $r - \bar{r}$ within the group, loss scaled by the constant $1/L$				
group size G	8				
updates per batch	1				
optimiser	SGD, constant learning rate 0.1, no warmup, no weight decay				
KL regularisation	none				
gradient clipping	1.0‡				
sampling	temperature 1, top-p 1, top-k off				
seeds	bank 43, verl.data 44 (training batch order)				
framework	verl 0.8.0 (`7aed6b2`), vLLM rollouts, bf16				

Table 1: Training and bank configuration. †MBPP+ outcomes are graded from the training rollouts of every step (Appendix B). The format gate scores a completion without an extractable code block 0 before the judge is asked. ‡Except 22 of the 126 Qwen2.5-1.5B runs, which trained unclipped (see text).

Runs are limited to 12 hours of wall-clock time. For Llama-3.1-8B's 200 steps at batch size 120, 9 of its 21 runs hit the limit early, between steps 107 and 148: all nine had drifted to completions near the 4096-token cap (mean length 2826-4065 tokens at their last step). We read the simulation prediction for all runs at each run's last validation step. Every run on the other four pairs reached within two steps of its budget.

Batch sizes are multiples of 3. Because a judge-rewarded run serves its judge on one of the node's four GPUs and trains on the other three, and verl's length-balanced batching requires the $8\times$batch sequences of a step to divide across them. Llama-3.1-8B's batch was raised from 48 to 120 after observing major training instabilities with batch size 48.

Gradient clipping. All runs use verl's default global gradient-norm clipping at 1.0, except 22 of the 126 Qwen2.5-1.5B runs, which were launched before we adopted this default and trained unclipped. Only one of the 22 unclipped runs ever exceeded it (once, peak 2.6). The simulation applies the same threshold to the norm of its update and never reaches it (largest simulated norm 0.07), so Eq. (6) is exact as written.

D.1 DATA AND PROMPTS

On MATH we use the training split of MATH-lighteval[2], restricted by difficulty level to match each model, adapting the per-model tiers of Zeng et al. (2025): levels 3-5 for both Qwen2.5 models and

[2] `https://huggingface.co/datasets/DigitalLearningGmbH/MATH-lighteval`

levels 1-2 for Llama-3.1-8B (they train Llama on GSM8K and level-1 MATH instead). Levels 3-5 leave 5586 problems, 4 of which are dropped for lacking a boxed reference answer. For levels 1-2 we have 1912 problems. The reference answer is the last `\boxed{}` of the reference solution. The filtered split is shuffled with a fixed seed and its first 1024 problems form both the simulation bank and the training set; the first 128 of those are the validation subset. On MBPP+[3] we consider all 378 problems except Mbpp/599, whose extended test suite runs for 19 s even on the reference solution. All 377 form both the bank and the training set. The policy and the judges see only the problem's base assertions and the extended EvalPlus tests the oracle runs are shown to neither.

Every policy is a base model and is never given a chat template. It only reads the rendered string below, with no role tokens or system prompt. The string is tokenized without special tokens, so no beginning-of-sequence token is prepended (Zeng et al. (2025), Shao et al. (2025)). Following Zeng et al. (2025), the weaker models read the less instruction-heavy Abel prompt (Chern et al., 2023) and Qwen2.5-7B the Qwen2.5-Math instruction (Yang et al., 2024). The MBPP prompt follows the format of Austin et al. (2021), showing the base assertions because they define the function's name and signature, with a closing instruction that fixes the output format for extraction.

```
[Qwen: Qwen2.5-7B]
{problem}
Please reason step by step, and put your final answer within \boxed{}.

[Abel: Qwen2.5-1.5B, Llama-3.1-8B]
Question:
{problem}
Answer:
Let's think step by step.

[MBPP: Qwen2.5-Coder-1.5B, Qwen2.5-Coder-7B]
You are an expert Python programmer.

{problem}

Your function must satisfy these tests:
{base assertions}

Return the complete function in a single ```python block.
```

The Abel prompt ends with a newline.

E Results Table

We report the results in Table 2.

F Additional Experiments

F.1 Varying bank size

The bank has P prompts with M completions each. For Qwen2.5-7B $\times$ MATH we use $P = |Q| = 1024$ and $M = 512$ in the main experiment. Figure 6 varies each on Qwen2.5-7B $\times$ MATH.

Fewer prompts. We simulate on disjoint prompt folds of the 1024×512 bank. At the early readout, P is the binding constraint. The final-step readout is uninformative on small banks, while the peak readout is nearly insensitive to P. Note that the sensitivity here depends on the dataset distribution.

More completions. We compare a separately generated 1024×4096 bank against its own eight disjoint chunks of 512 completions per prompt. On the shared shape it agrees with the main bank

[3] https://huggingface.co/datasets/evalplus/mbppplus

		simulator ρ			verifier statistics ρ			early RL
model	n	end	20	peak	J	prec	acc	ρ
MATH								
Qwen2.5-1.5B	126	+0.778	+0.659	**+0.787**	+0.607	+0.425	+0.165	+0.070
Qwen2.5-7B	126	+0.394	**+0.876**	+0.743	+0.638	+0.313	+0.569	+0.578
Llama-3.1-8B	21	+0.502	+0.559	+0.536	+0.363	+0.148	+0.126	**+0.561**
MBPP+								
Coder-1.5B	21	+0.533	**+0.573**	+0.527	−0.171	+0.524	+0.364	+0.458
Coder-7B	21	+0.873	**+0.874**	+0.865	−0.057	+0.799	+0.667	+0.525
mean		+0.616	**+0.708**	+0.692	+0.276	+0.442	+0.378	+0.438

Table 2: Spearman ρ between each predictor and the measured accuracy, over the n verifiers of each pair. Early RL is read after 20 training steps (26 on Llama-3.1-8B). Bold: best per row.

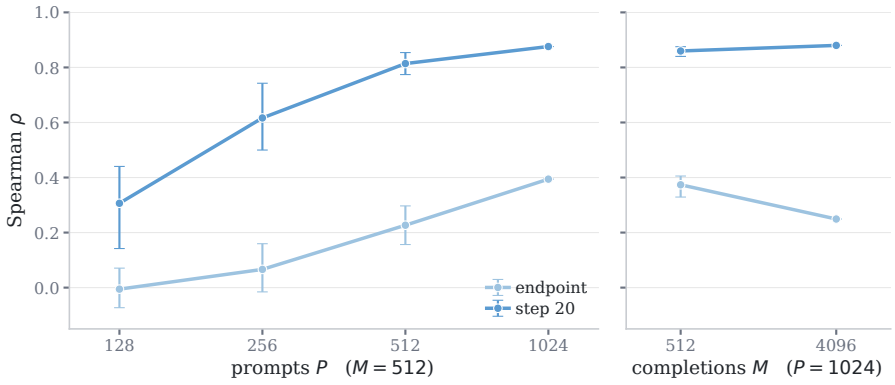


Figure 6: Prediction quality against bank shape, Qwen2.5-7B × MATH, 126 judges. Left: prompts P at $M = 512$, disjoint folds of the main 1024×512 bank. Right: completions M at $P = 1024$, the 1024×4096 bank against its eight disjoint 512-completion chunks.

within that noise. Eight times more completions slightly rises the early readout and worsen the final-step readout (Appendix F.2).

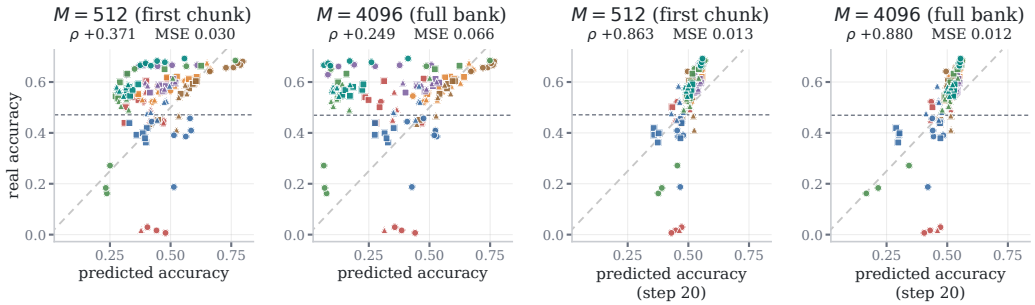


Figure 7: Simulated accuracy $\widehat{U}_T$ and $\widehat{U}_{20}$ against the real accuracy for different numbers of completions (Qwen2.5-7B/MATH).

Table 3: Effective number of completions per prompt of the simulated policy, $\exp(\bar{H})$ (perplexity), which starts at m. Median over verifiers. "end" is the last step of the training budget.

			step				
task	model	m	10	20	50	100	end
MATH	Qwen2.5-1.5B	256	230.5	114.8	9.8	4.7	2.6
	Qwen2.5-7B	512	396.6	225.9	31.4	7.4	2.7
	Llama-3.1-8B	256	20.0	11.9	3.8	1.9	1.3
MBPP+	Qwen2.5-Coder-1.5B	256	5.8	3.4	2.1	1.7	1.6
	Qwen2.5-Coder-7B	512	76.2	13.2	3.4	2.3	2.1

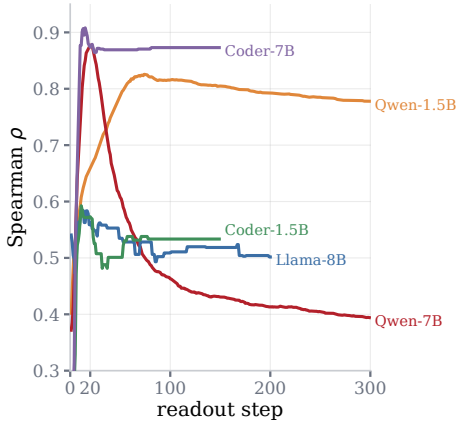


Figure 8: Spearman correlation between the simulation read at step t and the measured final accuracy. For Llama (step 200), Table 2 instead reads the 9 runs stopped by the time limit (steps 104–143) where they stopped.

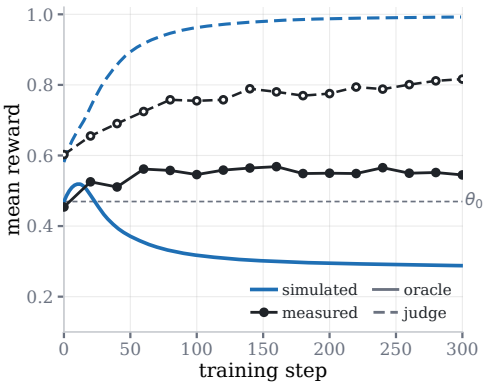


Figure 9: Qwen2.5-7B trained with the Gemma-3-12B judge (judge template 3, default threshold). Dotted: θ_0 accuracy.

F.2 Entropy collapse and readout step

Readout step. For every step t we read each verifier's predicted accuracy $\widehat{U}_t$ and compute its Spearman correlation with the measured final accuracy (Fig. 8). Averaged over the five pairs, the correlation peaks at step 17 and stays within 0.01 of that peak from step 15 to 22.

Concentration of the simulated policy. The simulated policy is a softmax over each prompt's m bank completions. Table 3 reports its perplexity $\exp(\bar{H})$, where $\bar{H}$ is the entropy over completions averaged over prompts. This is the effective number of completions per prompt, m at step 0. By the end, the median lies between 1.3 and 2.7 on every pair.

Under-predicted judge. On the endpoint readout prediction for Qwen2.5-7B, we can observe 53 verifiers ending more than 0.10 below their measured accuracy and 13 end more than 0.10 above. All 18 Gemma-3-12B verifiers are among the 53. Figure 9 shows one of them. In the simulation, accuracy rises from 0.470 to 0.519 at step 11 and then falls to 0.288, while the judge's reward approaches 0.99. In the real run, accuracy goes from 0.454 to 0.545 and the judge's reward reaches only 0.82.

F.3 Bank patching

The bank holds samples from π_0 only, while training moves the policy away from π_0. For Qwen2.5-7B on MATH, we test how much of the simulation's error is due to completions missing from the bank. To do so, we patch the bank with completions from late in each real run.

For each of the 126 verifiers, we build a patched bank from the verifier's training run.

- For each of the 1024 prompts, we take one of the 8 completions sampled for it at the last training step that drew it, chosen at random. These steps range from 211 to 300 (median 276) and are the same in every run.
- Each patching completion replaces a randomly chosen bank completion of the same prompt with the same oracle label. Every prompt keeps $m = 512$ completions, and the bank's initial accuracy is unchanged. A patched bank differs from the original in 1024 of its 524,288 completions.
- We compute $\nabla_\theta \log \pi_0(c \mid q)$ for each donor and sketch it with the same CountSketch as the bank.

Patched completions start with weight $1/m$ like every other completion, although π_0 would rarely sample them. We simulate each verifier on the bank patched by the corresponding post-RLVR samples, with the same settings as the unpatched simulation. Note that a patched bank contains completions from the run it predicts, so this is a diagnostic, not a predictor.

Result. Patching raises the Spearman correlation from 0.394 to 0.587 for the endpoint prediction and from 0.876 to 0.911 at step 20. The improvement is one-sided. Consider the verifiers whose unpatched endpoint misses the measured accuracy by more than 0.10. For the 13 over-predicted ones, the mean absolute error falls from 0.228 to 0.052. For the 53 under-predicted ones, it barely changes (0.215 to 0.197). Adding late completions thus removes most of the error where training ends worse than simulated, but not where the simulation collapses and training does not.

F.4 Drift of gradient geometry

The simulation keeps the kernel at θ_0 for the whole trajectory. For Qwen2.5-7B on MATH and a smaller sub-bank, we test how the predictions on a smaller sub change when the kernel is instead computed from the gradients at the end of training.

- We use the first 128 bank prompts, which are also the prompts on which the measured accuracy is evaluated, with 256 completions each (32,768 completions).
- We take the final weights θ_{300} of seven judge runs, chosen by hand They range from healthy training (final accuracy 0.66-0.67) over moderate final accuracy (0.58, 0.42) to collapse (0.16, 0.02).
- For each checkpoint, we recompute $\nabla_\theta \log \pi_{\theta_{300}}(c \mid q)$ for the same 32,768 completions and sketch it with the same CountSketch as the θ_0 kernel.
- We run the simulation for each verifier using the θ_{300} kernel based on that verifier's RLVR run in place of the θ_0 kernel. Everything else is unchanged: the completions, their labels and verdicts, the uniform initial policy over the completions, and the training settings. As a reference, we simulate RLVR for the same verifiers with the θ_0 kernel on the same completion sub-bank.

Result. For the endpoint readout, the simulation using the corresponding per-verifier post-RLVR θ_{300} kernels ranks the seven runs with a Spearman correlation of 0.964 (MSE 0.034). With the θ_0 kernel on the same small sub-bank, the correlation is 0.179 (MSE 0.070, Fig. 10). The θ_{300} predictions are more extreme than the measured outcomes. The three runs that end near 0.66 are predicted at 0.84 to 0.90, and the two Llama-judge runs that end at 0.42 and 0.16 are predicted at 0.10 and 0.06. For the step-20 readout, both kernels rank the seven runs identically (Spearman 0.929). With seven runs, these correlations are coarse.

F.5 Increasing the projection dimension

In practice, we compute X using gradient sketches with the CountSketch algorithm (Charikar et al., 2002), because full parameter gradients would be infeasible to store. This is sufficient because, in our kernel simulation, the model enters the simulated dynamics only through pairwise inner products of tangent features, which CountSketch preserves in expectation. Once the approximate kernel has been constructed, the simulation requires no further backward passes through the model.

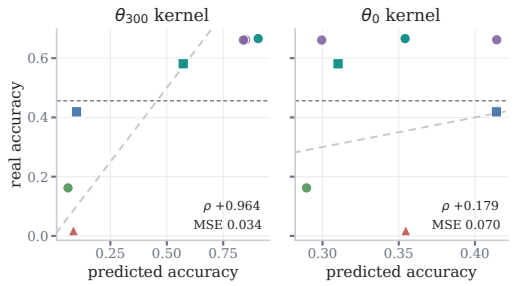


Figure 10: Seven Qwen2.5-7B runs on MATH, simulated on their first 128 prompts: endpoint prediction using the same verifier as in the corresponding training run, with the kernel recomputed using that run's final model weights (left) and with the θ_0 kernel (right), against the measured final accuracy. Horizontal line: initial accuracy.

All kernels use CountSketch with $k = 2048$ buckets per parameter group[4] of the model (giving for example 59392 as a sketch dimension in the case of Qwen2.5-7B). We construct two sketches, one at $k = 16384$ one at $k = 2048$ and simulate on both. The completions are the first 512 per prompt of the separately drawn 1024×4096 Qwen2.5-7B/MATH bank (Appendix F.1), so the two simulations differ only in the sketch width. The eight times wider sketch moves the endpoint correlation by 0.001, leaves the correlation at step 20 unchanged, and changes no verifier's predicted accuracy by more than 0.033.

[4]There is one parameter group per transformer layer and one group for all remaining parameters.