arXiv:2507.15778v2 [cs.CL] 15 May 2026

Stabilizing Knowledge, Promoting Reasoning: Dual-Token Constraints for RLVR

Jiakang Wang[2*], **Runze Liu**[1,2,3*], **Fuzheng Zhang**[2], **Xiu Li**[3],
Guorui Zhou[2], **Kun Gai**[2], **Ling Pan**[1]
[1]The Hong Kong University of Science and Technology, [2]Kuaishou Technology,
[3]Tsinghua University

Abstract

Reinforcement Learning with Verifiable Rewards (RLVR) has become an effective post-training method for improving the reasoning abilities of Large Language Models (LLMs). However, existing methods mainly apply uniform optimization constraints across all tokens, ignoring their heterogeneous roles. Prior work shows that high-entropy tokens are closely tied to reasoning, while low-entropy tokens primarily encode factual knowledge, and recent approaches attempt to exploit this distinction by isolating token updates via masking or asynchronous training. We argue that such isolation breaks the sequential dependency structure of autoregressive generation, leading to suboptimal learning. To address this, we propose **Archer**, an entropy-aware RLVR framework with **dual-token constraints** that preserves joint optimization while modulating update strength across token types. Our method introduces response-level entropy normalization for stable token classification and applies differentiated clipping ranges and KL regularization to encourage exploration on reasoning tokens while preserving knowledge tokens. Experiments on mathematical reasoning and code generation benchmarks show that Archer consistently outperforms strong baselines across multiple model scales, improving both *pass@1* and *pass@K* performance. These results highlight the importance of respecting sequence-level dependencies when designing fine-grained RL optimization strategies for LLMs.

1 Introduction

Large Language Models (LLMs) have shown strong capabilities across a wide range of domains, demonstrated by models like OpenAI's "o" series [32, 33] and DeepSeek-R1 [10]. While pre-training enables LLMs to obtain vast amount of world knowledge, post-training techniques such as Reinforcement Learning (RL) [10, 16, 45] and test-time scaling [39, 25] are crucial for enhancing their reasoning abilities. Compared to approaches like Monte Carlo Tree Search (MCTS) [41] and Process Reward Modeling [22, 42, 52], Reinforcement Learning with Verifiable Rewards (RLVR) has emerged as a simple yet effective way to further improve the reasoning abilities of LLMs [37, 48].

Recent studies have revealed that RL mainly improves reasoning by better integrating and organizing the model's existing abilities, such as reflection and planning, rather than directly changing the model's factual memory or basic skills (e.g., arithmetic) [9, 40, 19]. In particular, improvements are largely concentrated on tokens that mediate logical transitions (e.g., decision points and reasoning connectors) rather than tokens encoding factual knowledge. Prior analyses have shown that these reasoning-critical tokens tend to exhibit higher entropy, while knowledge-related tokens are typically low-entropy and more stable [43, 4]. This observation raises a fundamental question:

[*] Equal contribution

How should RL algorithms account for the heterogeneous functional roles of tokens during optimization?

Existing RLVR methods largely treat all tokens uniformly, applying identical update constraints across the entire response [37, 48]. To address this limitation, several recent works propose to explicitly differentiate token types based on entropy or related statistics, for example by masking gradients for low-entropy tokens or updating different token groups asynchronously [43, 8, 46]. While these approaches acknowledge token heterogeneity, they rely on *isolating* subsets of tokens during training. However, we argue that such isolation is fundamentally misaligned with the sequential nature of language modeling. In LLMs, tokens are generated autoregressively, and the optimization signal for a given token depends on its surrounding context. As a result, token-level updates are inherently **coupled across the sequence**. Completely masking or decoupling certain tokens disrupts this dependency structure, leading to suboptimal credit assignment and reduced learning efficiency for reasoning-critical steps.

To address these limitations, we introduce **Archer**, which adaptively constructs a token-conditioned trust region geometry while preserving joint optimization. Our key insight is that token heterogeneity should shape the permissible policy movement at each position, rather than by masking tokens or optimizing tokens uniformly. Locally, within each policy improvement step, trust regions should be enlarged for uncertain, reasoning-sensitive tokens, allowing the policy to flexibly explore alternative transitions, while being contracted for stable, knowledge-intensive positions to prevent large, destructive policy updates to reliable predictions. Globally, across training, reasoning-sensitive tokens should be allowed to drift farther from the reference policy to acquire reward-aligned reasoning behaviors, whereas knowledge-stable tokens should remain more tightly anchored to the pretrained distribution to preserve factual priors at the global level. Archer realizes these principles through a simple yet effective dual-anchor trust-region design. Instead of introducing auxiliary networks with additional computational overhead, we infer token roles by utilizing intra-response entropy quantiles zero-shot. Based on this information-theoretic proxy, we adaptively reshape the trust region along two complementary dimensions. The local principle is instantiated as a token-conditioned proximal constraint, which controls the permissible update magnitude at each position within a single optimization step through token-conditioned clipping. The global principle is instantiated as a token-conditioned reference-policy anchor, which controls the long-horizon drift of each position from the base model through token-conditioned KL regularization. Together, this dual-anchor mechanism effectively trades off between flexible reasoning and rigid factual preservation.

In summary, our contributions are as follows:

- We identify the limitations of token isolation strategies and entropy statistics in RLVR and introduce an intra-response entropy quantile scheme that infers token roles zero-shot.
- Archer, a token-conditioned dual-anchor trust-region framework that preserves joint autoregressive optimization while adaptively allocating policy movement budgets across tokens. Archer couples a local proximal anchor for step-wise update control with a global reference-policy anchor for long-horizon drift control.
- We demonstrate consistent improvements over strong baselines on both mathematical reasoning and code generation tasks with several model sizes.

2 Related Work

2.1 Reinforcement Learning for Large Language Models

Previous works have shown that RL, particularly Reinforcement Learning from Human Feedback (RLHF) [5, 24], is an effective tool for aligning LLMs with human preferences [34, 2]. With the recent success of scaling RL in LLMs [32, 10, 16], RLVR has emerged as an effective method to improve the reasoning ability of LLMs using rule-based rewards. However, approaches like GRPO [37] and its extensions [48, 26, 6, 50, 12] rely on response-level learning signals, which uniformly assign the same advantage value to all tokens within a response. This uniform treatment overlooks the distinct roles tokens play during reasoning (e.g., factual recall vs. logical inference), potentially leading to suboptimal learning at critical reasoning steps and limiting overall performance gains. Although process-based RL [15, 7, 51] and unsupervised RL [1, 4] provide fine-grained rewards for RL optimization, they still lack consideration for the functions of different tokens.

2.2 Critical Token Analysis in RL for Reasoning

Several recent studies have provided token-level analyses of RLVR training [46, 8, 43, 4]. Yang et al. [46] observe that low-probability tokens, often exhibiting high entropy, dominate the RL updates and the update of high-probability tokens are suppressed. Cui et al. [8] show that changes in policy entropy are linked to the covariance between action probabilities and advantages. Wang et al. [43] identify high-entropy tokens, referred to as "forking tokens", as logical connectors. Cheng et al. [4] further associate high-entropy tokens with reasoning-related behaviors, such as logical transitions and self-reflection. Unlike prior works that either completely isolate low-entropy tokens [43] or high-covariance tokens [43, 8], or train them separately [46], our approach employs joint training. While we similarly utilize entropy to distinguish between logic-oriented and knowledge-oriented tokens, we avoid direct filtering or separation. Instead, we apply differentiated training constraints, enabling us to preserve the capabilities of the base model while simultaneously encouraging more effective exploration during training.

3 Preliminaries

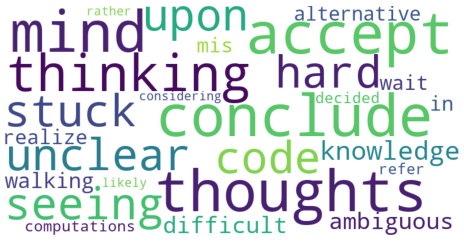
Group Relative Policy Optimization (GRPO) [37] proposes an alternative to the value-based advantage estimation used in Proximal Policy Optimization (PPO) [36]. Instead of learning a value model, GRPO estimates advantages by sampling multiple rollouts per prompt. Specifically, for a given prompt q, GRPO generates a group of responses $\{o^1, o^2, \ldots, o^G\}$ and computes the corresponding rewards $\{R^1, R^2, \ldots, R^G\}$. The advantage is then calculated as $\hat{A}_t^i = \frac{R^i - \text{mean}(\{R^i\}_{i=1}^G)}{\text{std}(\{R^i\}_{i=1}^G)}$. The GRPO loss is computed as:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim\mathcal{D},\{o^i\}_{i=1}^G\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o^i|}\sum_{t=1}^{|o^i|}\left(\min\left(r_t^i(\theta)\hat{A}_t^i, \text{clip}\left(r_t^i(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t^i\right) - \beta\mathbb{D}_{\text{KL}}(\pi_\theta\|\pi_{\text{ref}})\right)\right], \tag{1}$$

where $r_t^i = \frac{\pi_\theta(o_t^i|q,o_{<t}^i)}{\pi_{\theta_{\text{old}}}(o_t^i|q,o_{<t}^i)}$ denotes the importance sampling ratio, and β is a coefficient weighting the Kullback–Leibler (KL) divergence between the current policy π_θ and the reference policy π_{ref}.

4 Method

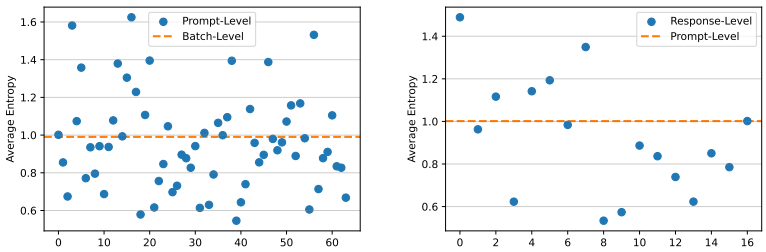
In this section, we introduce Archer, a novel RLVR approach with entropy-aware dual-token constraints. We begin by describing entropy-based method for identifying critical tokens (Section 4.1). Next, we discuss the limitations of prior methods in handling low-entropy tokens and motivate our approach for response-level entropy statistics (Section 4.1.1). Finally, we detail how Archer improves upon core constraints (clipping and KL) in previous RL algorithms by disentangling token-level optimization (Section 4.1.2).



(a) High-Entropy Tokens

(b) Low-Entropy Tokens

Figure 1: Word cloud visualization of a batch of responses: (a) High-entropy tokens; (b) Low-entropy tokens.



(a) Prompt-level vs. Batch-level (b) Response-level vs. Prompt-level

Figure 2: Comparison of average entropy: (a) Prompt-level vs. batch-level across all prompts; (b) Response-level vs. prompt-level across all responses.

4.1 Intra-Response Entropy as a Zero-Shot Proxy for Token Roles

Prior RL approaches like GRPO [37] and DAPO [48] typically adopt a uniform token-level optimization strength to all output tokens. This undifferentiated treatment fails to account for the distinct functional roles that different tokens play in the reasoning process (e.g., factual recall vs. logical decision points). Recent work shows that RL-driven improvements in LLM reasoning stem mainly from enhancing logical behaviors such as reflection and planning, which **integrate existing model capabilities**, rather than directly modifying the model's factual memory or primitive skills [49, 44]. Thus, during RL training, tokens associated with *factual knowledge* or *base-level skills* should largely retain their original distributions, while tokens involved in *logical reasoning and decision-making* require stronger learning signals and targeted exploration. Identifying these critical reasoning tokens is therefore a crucial first step. To address this issue, a crucial first step is to identify critical reasoning tokens.

Entropy-based Token Identification. Recent work proposes entropy as an effective signal for identifying critical tokens, observing that high-entropy tokens frequently appear at logical transition points between reasoning segments [43]. In contrast, low-entropy tokens typically complete ongoing statements or syntactic structures. This observation aligns with our hypothesis that entropy discriminates between reasoning-oriented and knowledge-oriented tokens. To empirically verify this, we analyze token entropy distributions of 1024 responses (each prompt 16 times) generated by DeepSeek-R1-Distill-Qwen-1.5B during training on mathematical tasks. Following Wang et al. [43], we visualize the top-100 highest entropy tokens and the top-100 lowest entropy tokens and retain tokens that appear more than 100 times. The visualization in Figure 1 shows that high-entropy tokens are mainly reasoning-related tokens, while most low-entropy tokens are related to factual knowledge or the suffix part of a word. These findings are also validated by recent studies [46, 4]. In summary, token entropy serves as an effective metric to distinguish between reasoning-oriented and knowledge-oriented tokens.

4.1.1 Response-Level Entropy Statistics

To distinguish token types, prior works compute token entropy quantiles or covariance statistics at the batch level [43, 8]. However, we find this suboptimal due to substantial entropy variation across responses from different prompts, as shown in Figure 2. For instance, some prompts yield responses with average entropy far above/below the batch mean (Figure 2 (a)); even within a single prompt, entropy can vary across sampled responses significantly (Figure 2 (b)).

Therefore, batch-level statistics for token classification introduce a key drawback: if a response's overall entropy is low, even critical reasoning tokens may be misclassified as low-entropy, resulting in effective training. For example, using the 80th percentile as a threshold can result in only 4.34% of tokens being labeled as high-entropy in low-entropy responses. Conversely, for high-entropy responses, the proportion of high-entropy tokens may be abnormally inflated. To mitigate this, we adopt a **response-level** entropy statistics method for token classification, computing entropy quantiles independently within each response. Given a batch of N rollout responses, let e_t^i be the entropy of

token t in response o^i. We compute the ρ-quantile of token entropy for each response as a threshold:

$$\tau_\rho^i = \text{Quantile}\left(\{e_t^i\}_{t=1}^{|o^i|}, \rho\right), \tag{2}$$

where $\rho \in (0, 1)$ denotes the quantile level (e.g., $\rho = 0.8$ corresponds to the 80th percentile).

4.1.2 Dual-Anchor Token-Conditioned Trust Region

To address these issues, we propose a framework that performs synchronous updates while applying differentiated training constraints to different token types. Using response-level entropy as the criterion, we distinguish knowledge-type (low-entropy) from reasoning-type (high-entropy) tokens. Unlike prior works that adopt isolation strategies (e.g., gradient masking or asynchronous training), our method optimizes all tokens jointly but dynamically modulates the constraint intensity. We first introduce a token-level gating variable $\omega_t^i = \mathbb{I}(e_t^i \geq \tau_\rho^i) \in \{0, 1\}$ based on the entropy quantile τ_ρ^i computed with Eq. (2).

Local Proximal Anchor. To control the magnitude of policy updates at each step, we apply stricter clip ranges to knowledge-type (low-entropy) tokens to preserve the base model's capabilities and looser clip ranges to reasoning-type (high-entropy) tokens to encourage exploratory behavior. Given a batch of responses, we first compute the entropy quantile τ_ρ^i of token entropy within each response using (2). Based on the computed entropy threshold, we categorize tokens into different types and assign distinct clipping ranges to each type accordingly:

$$\varepsilon(e_t^i) = \omega_t^i \varepsilon^{\text{r}} + (1 - \omega_t^i)\varepsilon^{\text{k}} \tag{3}$$

Global Reference-Policy Anchor. In RL training, the KL divergence penalty is commonly used to constrain the overall deviation of the trained policy from a reference policy [37]. Although recent works [48, 26, 13, 6, 50, 12] advocate removing the KL divergence penalty, ProRL [23] argues that this typically holds for base models without extensive SFT and using the KL penalty is crucial for training stability. Our experimental results also confirm that fully removing the KL penalty leads to training collapse and degraded performance, as shown in Section 5.3.2. Moreover, applying uniform KL penalties across all tokens, including high-entropy ones, significantly slows learning and reduces final performance.

Therefore, we extend the conventional KL penalty by adapting it based on the functional type of each token. Specifically, we apply a stronger KL penalty (i.e., a larger KL weight) to knowledge-type tokens (low entropy) to preserve the base model's factual knowledge. In contrast, we apply a weaker KL penalty (i.e., a smaller KL weight) to reasoning-type tokens (high entropy), enabling greater flexibility in critical reasoning regions. The coefficients of KL constraints are computed as:

$$\beta(e_t^i) = \omega_t^i \beta^{\text{r}} + (1 - \omega_t^i)\beta^{\text{k}} \tag{4}$$

Finally, the overall objective of our algorithm is formulated as follows:

$$\begin{aligned}\mathcal{J}_{\text{Archer}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D},\{o^i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)}\Bigg[\frac{1}{\sum_{i=1}^{G}|o^i|}\sum_{i=1}^{G}\sum_{t=1}^{|o^i|} \\ \left(\min\left(r_t^i(\theta)\hat{A}_t^i, \text{clip}\left(r_t^i(\theta), 1-\varepsilon(e_t^i), 1+\varepsilon(e_t^i)\right)\hat{A}_t^i\right) - \beta(e_t^i)\mathbb{D}_{\text{KL}}(\pi_\theta\|\pi_{\text{ref}})\right)\Bigg],\end{aligned} \tag{5}$$

where differentiated clipping and KL constraints are denoted using red color. The full algorithm of Archer is shown in Algorithm 1.

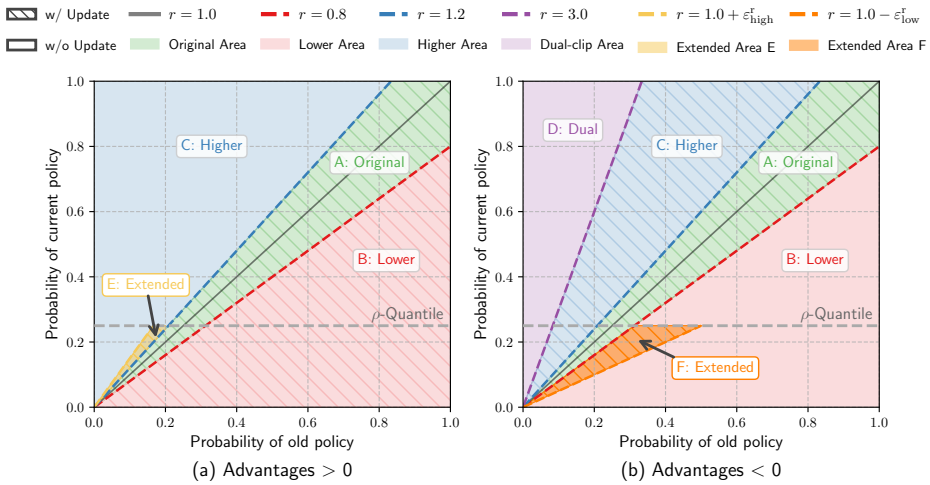
4.1.3 Visualization of RL Optimization Regions

To better clarify the mechanism of our method, we visualize the optimization regions produced by the GRPO loss for different token types in Figure 3. Each data point in the coordinate system represents the importance sampling ratio r_t^i between the current and old policy probabilities. Figure 3(a) shows tokens with positive advantage values ($\hat{A}_t^i > 0$), while Figure 3(b) shows tokens with negative advantages ($\hat{A}_t^i < 0$). The colored regions mark the areas divided by the clipping thresholds. The

Algorithm 1 Archer

Input: Base model π_{base}, prompt dataset $\mathcal{D}$, quantile level ρ, clipping thresholds $\varepsilon^{\text{r}}, \varepsilon^{\text{k}}$, KL coefficients $\beta^{\text{r}}, \beta^{\text{k}}$

Initialize policy model $\pi_\theta \leftarrow \pi_{\text{base}}$ and reference model $\pi_{\text{ref}} \leftarrow \pi_{\text{base}}$
for step $= 1, 2, \ldots, T$ **do**
  Sample a batch of prompts $\mathcal{D}_b$ from $\mathcal{D}$
  Generate responses $\{o^i\}_{i=1}^{G}$ for each prompt q in the batch
  for each response $|o^i|$ **do**
    Compute the ρ-quantile of token entropy τ_ρ^i with (2)
    Compute clipping thresholds and coefficients of KL penalty with (3) and (4)
  end for
  Update the policy model π_θ using (5)
end for



(a) Advantages > 0 (b) Advantages < 0

Figure 3: Visualization of PPO clip regions. The x-axis shows the sampled probability of a specific token $\pi_{\theta_{\text{old}}}$ during generation, and the y-axis shows the probability of the token under the current policy π_θ. Region A denotes the optimization area for original GRPO. Regions B and C are areas below and above the clipping threshold, respectively. Region D is the area for dual-clip [47]. (a) When $\hat{A}_t^i > 0$, Archer optimizes region E. (b) When $\hat{A}_t^i < 0$, Archer optimizes region F.

shaded areas (Regions A, B for $\hat{A}_t^i > 0$ and Regions A, C for $\hat{A}_t^i < 0$) indicate where GRPO updates the model. Our method **extends the clipping boundaries for high-entropy tokens**, which are typically low-probability but are important for reasoning. As shown in Figure 3, Regions E and F correspond to the **newly extended optimization areas** introduced by Archer. Region E provides **additional reward signals** to high-entropy tokens when $\hat{A}_t^i > 0$, while Region F applies **stronger penalties** to high-entropy tokens when $\hat{A}_t^i < 0$. This design **increases the model's focus on learning reasoning-critical tokens**.

5 Experiments

5.1 Setup

Models and Baselines. We adopt DeepSeek-R1-Distill-Qwen-1.5B [10], Qwen3-4B, Qwen3-8B, and Qwen3-30B-A3B[1] [45] as the base model and compare Archer against the following methods: (1) **Base Model**: The raw distilled model without further training. (2) **GRPO** [37]: A common RLVR algorithm combined with token-level loss [48]. For DeepSeek-R1-Distill-Qwen-1.5B, we additionally compare: (3) **DeepScaleR-1.5B** [28]: A 1.5B model trained on mathematical tasks with iterative

[1]We use non-thinking mode for Qwen3 models.

context length expansion. (4) **DeepCoder-1.5B** [27]: A 1.5B model trained on code datasets, also utilizing context expansion strategies. (5) **Nemotron-1.5B** [23]: Currently the best 1.5B reasoning model that RL-trained with DeepSeek-R1-Distill-Qwen-1.5B as the base model. For Qwen3-4B and Qwen3-8B, we also compare: (6) **80/20** [43] and (7) **AR-Lopti** [46].

Training Data. For code domain, we construct a high-quality code training dataset from three publicly available sources: DeepCoder [27], CodeContests [21], and CodeForces [35]. Notably, CodeContests and CodeForces augment original problems with extensive test cases, which reduces false positives (i.e., incorrect solutions that pass test cases). Therefore, we prioritize these two datasets over DeepCoder in cases of duplication. After rigorous cleaning and filtering steps (detailed in Appendix A.1), we obtain a final corpus of 6,753 programming problems. For mathematics domain, we use datasets from DeepScaleR [28], Skywork-OR1 [12], and DAPO [48]. We merge these datasets and apply N-gram overlap removal to eliminate duplicates. After additional verification and filtering steps (see Appendix A.1), we derive a final mathematics training set of 51,800 problems.

Evaluation and Metrics. We conduct evaluation on both mathematical and coding benchmarks. For mathematics, we use six challenging datasets: AIME24 [30], AIME25 [31], AMC23 [29], MATH-500 [22], Minerva Math [18], and OlympiadBench [11]. For coding, we adopt the widely used LiveCodeBench v5 (2024.08.01-2025.02.01) and v6 (2025.02.01-2025.05.01) [14], which emphasize reasoning-intensive code generation. We use `vLLM` [17] with temperature set to 0.8, `top_p` set to 1.0, and maximum output length set to 32,768 tokens for inference. Due to the high variance of the outputs from reasoning models, we report *avg@K* (pass@1 performance averaged over K outputs) and *pass@K* for each benchmark. For benchmarks with few samples (AIME24/25 and AMC23), we set a larger K=64. We use K=16 for LiveCodeBench v6, K=8 for LiveCodeBench v5 and Minerva, and K=4 for MATH-500 and OlympiadBench. To ensure accurate evaluation, we adopt the verification functions from both DeepScaleR and Math-Verify[2] for mathematics problems.

Implementation Details. We perform RL training using the `verl` framework [38]. For GRPO-based baselines, we use clipping thresholds of $\varepsilon_{\text{low}} = 0.2$ and $\varepsilon_{\text{high}} = 0.28$. KL penalty loss and entropy regularization loss are omitted from the loss function. During training, we sample 16 rollouts per prompt with a temperature of 1.0. We set a maximum response length of 32,768 for DeepSeek-R1-Distill-Qwen-1.5B, 8,192 for Qwen3-4B and Qwen3-30B-A3B, and 6,144 for Qwen3-8B. The batch size is set to 64, the mini-batch size to 32, and the learning rate to 1×10^{-6}. For Archer, we set $\rho = 0.8$ following Wang et al. [43]. For clipping ranges and KL coefficients, we use $\varepsilon^{\text{r}} = 0.5$, $\varepsilon^{\text{k}} = 0.2$, $\beta^{\text{r}} = 0.0$, and $\beta^{\text{k}} = 0.001$. All experiments are conducted on 2 compute nodes, each equipped with 8 $\times$ NVIDIA H800 80GB GPUs.

5.2 Main Results

Comparison with Base Model and GRPO. The results in Table 1 and 2 show that our dual-token constraint training strategy leads to significant improvements on both mathematical and coding tasks. Compared to the original base model (DeepSeek-R1-Distill-Qwen-1.5B), the average accuracy increases by **18.1%** on AIME24 and **10.3%** on AIME25, resulting in an average gain of **12.3%**. On coding benchmarks, the accuracy rises by **12.7%** on LiveCodeBench v5 and **13.0%** on LiveCodeBench v6. When applying our method upon GRPO, the performance consistently exceeds that of GRPO across all benchmarks, with average gains of **5.6%** and **3.0%** for mathematical and coding tasks, respectively. When applying Archer to Qwen3-4B and Qwen3-8B, Archer still outperforms GRPO by a large margin. These results demonstrate the effectiveness of our optimization approach.

Comparison with SOTA Reasoning Models. We also compare Archer with SOTA reasoning models trained with RL using DeepSeek-R1-Distill-Qwen-1.5B as the base model. For coding tasks, our approach outperforms all comparable models, including the programming-specialized **DeepCoder-1.5B** and the general-purpose **Nemotron-1.5B**. On mathematical reasoning, our model achieves the highest average accuracy, surpassing both math-specialized models (**DeepScaleR-1.5B**) and **Nemotron-1.5B**. We report the training costs of Archer and these open-source reasoning models, including the number of training steps, stages, and GPU hours in Table 6. Notably, our model

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

Table 1: Evaluation results on mathematical benchmarks. The results of Archer are shaded and the highest values are **bolded**.

Method	AIME24		AIME25		AMC23		MATH-500		Minerva		Olympiad		Avg.
	avg@64	pass@64	avg@64	pass@64	avg@64	pass@64	avg@4	pass@4	avg@8	pass@8	avg@4	pass@4	
DeepSeek-R1-1.5B	30.6	80.0	23.5	63.3	70.7	**100.0**	83.6	92.4	27.6	48.2	44.6	59.4	46.8
↳ GRPO	42.1	80.0	28.6	56.7	80.3	97.5	87.6	**94.6**	29.2	46.3	53.2	65.8	53.5
↳ DeepScaleR-1.5B	42.0	**83.3**	29.0	63.3	81.3	**100.0**	87.7	93.6	30.3	**51.1**	50.7	61.0	53.5
↳ Nemotron-1.5B	48.0	76.7	33.1	60.0	86.1	97.5	90.6	93.6	35.3	47.8	59.2	66.8	58.7
↳ Archer-Math-1.5B	**48.7**	**83.3**	**33.8**	**70.0**	86.0	97.5	**90.8**	94.4	**35.7**	**51.1**	**59.3**	**67.1**	**59.1**
Qwen3-4B	23.6	56.7	18.3	63.3	67.7	95.0	84.5	92.4	41.5	56.3	54.1	66.6	48.3
↳ GRPO	43.4	**83.3**	35.5	**70.0**	84.3	**97.5**	91.7	95.8	47.2	58.5	67.4	75.8	61.6
↳ 80/20	50.4	**83.3**	40.5	76.7	88.9	**97.5**	93.6	97.2	47.2	57.0	69.3	78.0	65.0
↳ AR-Lopti	48.6	80.0	38.9	76.7	88.7	**97.5**	93.7	**97.4**	49.1	58.5	69.5	78.2	64.8
↳ Archer-Math-4B	**51.4**	**83.3**	**43.1**	**70.0**	**91.0**	**97.5**	**95.1**	**97.4**	**51.2**	**60.7**	**71.6**	**79.4**	**67.1**
Qwen3-8B	27.0	63.3	19.1	56.7	68.9	**97.5**	83.6	92.4	43.9	58.1	55.7	69.6	49.7
↳ GRPO	50.3	**83.3**	34.1	66.7	84.1	95.0	92.7	96.0	50.2	61.4	68.2	76.1	63.3
↳ 80/20	50.9	80.0	40.1	70.0	87.3	**97.5**	94.1	96.8	49.2	59.6	70.0	78.5	65.3
↳ AR-Lopti	50.6	**83.3**	38.1	60.0	88.4	**97.5**	93.3	97.4	49.6	58.8	68.9	78.3	64.8
↳ Archer-Math-8B	**56.1**	76.7	**43.6**	**73.3**	**91.8**	**97.5**	**94.8**	**97.8**	**51.5**	**62.1**	**69.4**	**77.3**	**67.9**
Qwen3-30B-A3B	30.1	70.0	19.9	56.7	74.3	**100.0**	88.4	96.0	47.7	59.6	59.6	71.2	53.3
↳ GRPO	59.5	80.0	43.1	73.3	91.9	97.5	95.3	97.8	51.9	61.8	71.0	79.2	68.8
↳ Archer-Math-30B	**70.3**	**90.0**	**53.9**	**86.7**	**94.4**	97.5	**96.4**	**99.0**	**55.5**	**64.7**	**77.4**	**84.0**	**74.6**

Table 2: Evaluation results on code benchmarks. The results of Archer are shaded and the highest values are **bolded**.

Method	LCB v5 (2024.08.01-2025.02.01)		LCB v6 (2025.02.01-2025.05.01)		Avg.
	avg@8	pass@8	avg@16	pass@16	
DeepSeek-R1-1.5B	16.7	29.0	17.2	34.4	17.0
↳ GRPO	26.0	40.5	27.6	43.5	26.8
↳ DeepCoder-1.5B	23.3	39.1	22.6	42.0	23.0
↳ Nemotron-1.5B	26.1	35.5	29.5	42.8	27.8
↳ Archer-Code-1.5B	**29.4**	**43.7**	**30.2**	**45.8**	**29.8**
Qwen3-4B	23.8	35.8	24.0	35.1	23.9
↳ GRPO	40.8	55.2	36.6	45.0	38.7
↳ Archer-Code-4B	**42.1**	**57.0**	**37.5**	**48.1**	**39.8**

achieves the best results with only **single-stage** training and **fewer** GPU hours, without the complex multi-round training used by the other methods. In addition to improvements in **pass@1**, our model also shows advantages in **pass@K** metrics, which suggests stronger reasoning diversity and higher capability limits of our method.

5.3 Analysis

5.3.1 Impact of Different Entropy Thresholds

We further ablate the entropy quantile threshold ρ used to identify high-entropy tokens in Eq. (2). A larger ρ selects fewer tokens as reasoning-type tokens, making the criterion more conservative, while a smaller ρ allows more tokens to receive relaxed clipping and weaker KL constraints.

Table 3: Ablation of the entropy quantile threshold ρ on mathematical benchmarks. The highest values are **bolded**.

ρ	AIME24	AIME25	AMC23	MATH-500	Minerva	Olympiad	Avg.
0.7	50.7	42.6	**91.2**	**95.3**	50.6	71.0	66.9
0.8	**51.4**	**43.1**	91.0	95.1	**51.2**	**71.6**	**67.1**
0.9	48.3	39.9	90.9	95.0	50.4	70.3	65.8

As shown in Table 3, $\rho = 0.8$ achieves the best average performance, while the stricter threshold $\rho = 0.9$ consistently degrades results. This suggests that an overly conservative threshold misclassifies

many potentially reasoning-critical tokens as low-entropy tokens, thereby restricting their updates with tighter clipping and stronger KL regularization. In contrast, using a moderately smaller threshold exposes more high-entropy tokens to exploratory updates, which provides richer learning signals and leads to better model performance.

5.3.2 Impact of Different KL Weights

We next vary the KL weight on low-entropy tokens and use the average n-gram repetition ratio as a proxy for collapse. Table 4 and Figure 7 show that both removing KL regularization and using an overly large weight hurt performance. Without KL, entropy drops quickly and repetition rises, leading to unstable training and lower final accuracy. With a large KL weight, the model better preserves the base policy but learns more slowly.

Table 4: LiveCodeBench v5 performance with varying KL weights on low-entropy tokens.

KL Weight	LiveCodeBench v5 (avg@8)
0.0	26.6
0.001	29.4
0.005	26.2

In summary, both too little and too much KL regularization hurt the final model quality. Insufficient weighting accelerates learning but makes collapse more likely, which ends up reducing performance. In contrast, excessive weighting limits learning on low-entropy tokens and thus restricts the model's capabilities. These results **highlight the need for KL regularization on low-entropy tokens** to keep the model close to the base policy, which helps prevent collapse and retain key abilities. These observations further support our view that low-entropy tokens should be included in training, as masking them negatively affects overall learning.

5.3.3 Impact of Clip Ranges on Different Token Types

We introduce different clip thresholds for different token types in (3). To investigate how the thresholds influence model performance, we vary the clip ranges for both high-entropy (ε^{r}) and low-entropy tokens (ε^{k}) and the results are shown in Table 5, Figure 8, and 9.

Table 5: LiveCodeBench v5 performance under different low-/high-entropy token clip thresholds.

ε^{k}	ε^{r}	LiveCodeBench v5 (avg@8)
Varying Low-Entropy Token Clip		
0.1	0.4	24.6
0.2	0.4	**28.7**
0.3	0.4	26.0
Varying High-Entropy Token Clip		
0.2	0.2	27.7
0.2	0.4	28.7
0.2	0.5	**29.4**
0.2	0.6	26.0

Different Low-Entropy Token Clip Thresholds. As shown in Figure 8, we observe that increasing the clip threshold for low-entropy tokens produces effects similar to reducing their KL penalty weight: the model's entropy decreases more rapidly, which leads to faster learning and earlier performance improvements. However, this also causes the repetition ratio to rise more quickly, making the model more susceptible to overfitting or collapse, which harms final performance.

On the other hand, lowering the clip threshold for low-entropy tokens has effects similar to increasing their KL weight: improvements on LiveCodeBench v5 are slower and tend to converge a lower level. Interestingly, we observe an counterintuitive entropy dynamic during training. Instead of a consistently slow decline, as seen with higher KL weights, entropy initially drops sharply, then plateaus and remains stable.

These results indicate that adjusting the clip threshold for low-entropy tokens strongly affects both the training process and the final model performance. In contrast, the model is much less sensitive to changes in the clip threshold for high-entropy tokens.

Different High-Entropy Token Clip Thresholds. As illustrated in Figure 9, increasing the clip threshold for high-entropy tokens encourages more exploration in the model's reasoning. This leads to a slightly faster reduction in entropy during training and can improve the performance. However, these differences become more noticeable mainly in the later stages of training. In the early stages, training dynamics and LiveCodeBench v5 performance show little difference across various high-entropy clip values.

6 Conclusion

In this work, we propose an entropy-aware, synchronized training framework that updates all tokens simultaneously while applying different regularization and clipping strategies depending on the type of token. By encouraging exploration on reasoning-related tokens and preserving factual correctness for knowledge-related tokens, our method balances the goals of keeping factual accuracy and improving logical reasoning. Extensive experiments on mathematical and code reasoning benchmarks show that our approach improves over the base model and outperforms existing SOTA models. These results indicate that coordinating the learning processes of different token types through entropy-aware constraints improves the reasoning abilities of LLMs. We believe this work highlights the interaction between factual knowledge and reasoning processes during RL training of LLMs, and suggests future research directions for fine-grained optimization methods that respect the inherent structural dependencies in natural language.

Limitations. Although Archer significantly outperforms GRPO, it still has several limitations. First, entropy is only a heuristic proxy for token roles. Second, our current method uses a binary token partition with fixed clipping and KL coefficients. Future work could explore continuous constraints or more fine-grained token types.

References

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

[2] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[3] Yang Chen, Zhuolin Yang, Zihan Liu, Chankyu Lee, Peng Xu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron: Advancing math and code reasoning through reinforcement learning. *arXiv preprint arXiv:2505.16400*, 2025.

[4] Daixuan Cheng, Shaohan Huang, Xuekai Zhu, Bo Dai, Wayne Xin Zhao, Zhenliang Zhang, and Furu Wei. Reasoning with exploration: An entropy perspective. *arXiv preprint arXiv:2506.14758*, 2025.

[5] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf.

[6] Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. Gpg: A simple and strong reinforcement learning baseline for model reasoning. *arXiv preprint arXiv:2504.02546*, 2025.

[7] Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

[8] Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng, Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and Ning Ding. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

[9] Kanishk Gandhi, Ayush K Chakravarthy, Anikait Singh, Nathan Lile, and Noah Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STars. In *Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=QGJ9ttXLTy.

[10] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[11] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

[12] Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, et al. Skywork open reasoner 1 technical report. *arXiv preprint arXiv:2505.22312*, 2025.

[13] 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*, 2025.

[14] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

[15] Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Refining credit assignment in RL training of LLMs. In *International Conference on Machine Learning (ICML)*, 2025. URL https://openreview.net/forum?id=Myx2kJFzAn.

[16] 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.

[17] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[18] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving Quantitative Reasoning Problems with Language Models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 3843–3857. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/18abbeef8cfe9203fdf9053c9c4fe191-Paper-Conference.pdf.

[19] Dacheng Li, Shiyi Cao, Tyler Griggs, Shu Liu, Xiangxi Mo, Eric Tang, Sumanth Hegde, Kourosh Hakhamaneshi, Shishir G Patil, Matei Zaharia, et al. Llms can easily learn to reason from demonstrations structure, not content, is what matters! *arXiv preprint arXiv:2502.07374*, 2025.

[20] Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. NuminaMath. [https://github.com/p

roject-numina/aimo-progress-prize](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024. Accessed: 2026-04-30.

[21] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://www.science.org/doi/abs/10.1126/science.abq1158.

[22] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

[23] Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. ProRL: Prolonged reinforcement learning expands reasoning boundaries in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://openreview.net/forum?id=YPsJha5HXQ.

[24] Runze Liu, Fengshuo Bai, Yali Du, and Yaodong Yang. Meta-reward-net: Implicitly differentiable reward learning for preference-based reinforcement learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 22270–22284. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/8be9c134bb193d8bd3827d4df8488228-Paper-Conference.pdf.

[25] Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. *arXiv preprint arXiv:2502.06703*, 2025.

[26] 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.

[27] Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, 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, 2025. Notion Blog. Accessed: 2026-04-30.

[28] Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2, 2025. Notion Blog. Accessed: 2026-04-30.

[29] MAA. American mathematics contest 12 (amc 12), November 2023. URL https://artofproblemsolving.com/wiki/index.php/AMC_12_Problems_and_Solutions. Accessed: 2026-04-30.

[30] MAA. American invitational mathematics examination (aime), February 2024. URL https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. Accessed: 2026-04-30.

[31] MAA. American invitational mathematics examination (aime), February 2025. URL https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. Accessed: 2026-04-30.

[32] OpenAI. Learning to reason with llms, 2024. URL https://openai.com/index/learning-to-reason-with-llms. Accessed: 2026-04-30.

[33] OpenAI. Openai o3-mini, 2024. URL https://openai.com/index/openai-o3-mini. Accessed: 2026-04-30.

[34] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

[35] Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. Codeforces. https://huggingface.co/datasets/open-r1/codeforces, 2025. Accessed: 2026-04-30.

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

[37] 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.

[38] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

[39] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

[40] Jean Vassoyan, Nathanaël Beau, and Roman Plaud. Ignore the kl penalty! boosting exploration on critical tokens to enhance rl fine-tuning. *arXiv preprint arXiv:2502.06533*, 2025.

[41] Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus Mcaleer, Ying Wen, Weinan Zhang, and Jun Wang. AlphaZero-like tree-search can guide large language model decoding and training. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 49890–49920. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/wan24c.html.

[42] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, 2024.

[43] Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xiong-Hui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://openreview.net/forum?id=yfcpdY4gMP.

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

[45] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin

Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[46] Zhihe Yang, Xufang Luo, Zilong Wang, Dongqi Han, Zhiyuan He, Dongsheng Li, and Yunjian Xu. Do not let low-probability tokens over-dominate in RL for LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=FOnAdLo0tM.

[47] Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, Qiaobo Chen, Yinyuting Yin, Hao Zhang, Tengfei Shi, Liang Wang, Qiang Fu, Wei Yang, and Lanxiao Huang. Mastering complex control in moba games with deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):6672–6679, Apr. 2020. doi: 10.1609/aaai.v34i04.6144. URL https://ojs.aaai.org/index.php/AAAI/article/view/6144.

[48] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://openreview.net/forum?id=2a36EMSSTp.

[49] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://openreview.net/forum?id=4OsgYD7em5.

[50] Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, Xiangpeng Wei, Xiangyu Yu, Gaohong Liu, Juncai Liu, Lingjun Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Ru Zhang, Xin Liu, Mingxuan Wang, Yonghui Wu, and Lin Yan. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*, 2025.

[51] Kaiwen Zha, Zhengqi Gao, Maohao Shen, Zhang-Wei Hong, Duane S Boning, and Dina Katabi. Rl tango: Reinforcing generator and verifier together for language reasoning. *arXiv preprint arXiv:2505.15034*, 2025.

[52] Jian Zhao, Runze Liu, Kaiyan Zhang, Zhimu Zhou, Junqi Gao, Dong Li, Jiafei Lyu, Zhouyi Qian, Biqing Qi, Xiu Li, et al. Genprm: Scaling test-time compute of process reward models via generative reasoning. *arXiv preprint arXiv:2504.00891*, 2025.

A Experimental Details

A.1 Dataset

A.1.1 Code Domain

Data Sources and Integration. The code dataset is compiled from three publicly available sources: DeepCoder, CodeContests, and CodeForces. Notably, CodeContests and CodeForces extend their original problem sets with a larger number of test cases, improving the reliability of evaluation and reducing the incidence of false positives—i.e., incorrect code that inadvertently passes tests. As such, these two datasets are prioritized. In cases of duplication with DeepCoder, we retain the entries from either CodeContests or CodeForces.

Data Cleaning and Filtering Pipeline. We apply a rigorous multi-stage cleaning and selection process to ensure dataset quality:

1. **Test Case Preprocessing:** We remove illustrative test cases embedded in problem descriptions and discard problems with fewer than five test cases, which are more susceptible to false positives.
2. **Model Validation and Difficulty Filtering:** Each problem is evaluated using 8-sample generation with a strong language model (Qwen3-30B-A3B [45]). We exclude problems for which all samples fail verification, filtering out flawed questions (e.g., with invalid test cases), overly long I/O problems beyond the verifier's capacity, or those that are excessively difficult—even for strong models. This reduces potential false negatives.
3. **Problem Deduplication:** We perform N-gram-level deduplication to eliminate duplicate questions within the training corpus.
4. **Test Set Contamination Prevention:** To prevent data leakage, we remove any overlapping problems by conducting N-gram-level deduplication against the evaluation set of LiveCodeBench v5.
5. **Sampling Stability Filtering:** Using a warm-start model (DeepSeek-R1-Distill-Qwen-1.5B), we generate 8 additional samples per problem. We remove problems where all generations are either completely correct or completely incorrect, thereby ensuring sufficient learning signal and gradient diversity.

Data Standardization. All retained code problems are reformatted into either *function-call* or *stdin/stdout* formats, enabling consistent and automated validation via a code verifier.

Final Dataset. Following the aforementioned pipeline, we construct a high-quality code training dataset consisting of **6,753** problems.

A.1.2 Mathematics Domain

Data Sources and Integration. For the mathematics domain, we leverage existing curated datasets rather than raw symbolic corpora such as NuminaMath [20]. Specifically, we integrate three high-quality, verifiable datasets: DeepScaleR, Skywork-OR1, and DAPO. The datasets are merged and deduplicated using N-gram overlap removal to eliminate redundancy.

Data Cleaning and Filtering Pipeline.

1. **Model Validation and Filtering:** Each math problem undergoes 8-sample generation using the Qwen3-30B-A3B model, followed by verification using a mathematical logic verifier. Problems for which all samples fail are excluded to remove noise, overly complex items, or verification bottlenecks that might cause false negatives.
2. **Sampling Stability Filtering:** We repeat the 8-sample generation process using a warm-start model (DeepSeek-R1-Distill-Qwen-1.5B) and discard problems with homogeneous sampling outcomes (i.e., all correct or all incorrect).

3. **Test Set Contamination Prevention:** To avoid contamination of evaluation benchmarks, we perform N-gram deduplication against the AMC competition datasets (AIME24 and AIME25), ensuring zero overlap.

Final Dataset. After rigorous verification and filtering, we obtain a final mathematics training corpus comprising approximately **51,800** high-quality problems suitable for reinforcement learning.

B Additional Experimental Results

B.1 Efficiency

Table 6: Computational efficiency comparison between Archer and the baselines on 1.5B models.

Method	Training Steps	Stages	GPU Hours
	Math RL		
DeepScaleR-1.5B	1750	3	3,800 A100
Nemotron-1.5B	2500	8	16,000 H100
Archer-Math-1.5B	520	1	1,900 H800
	Code RL		
DeepCoder-1.5B	—	—	—
Nemotron-1.5B	2500	8	16,000 H100
Archer-Code-1.5B	320	1	1,000 H800

B.2 Training Dynamics

We provide training dynamics curves and test curves on mathematical tasks in Figure 4, 5, and 6. Figure 4 and 5 show that Archer outperforms GRPO with lower repetition rate.

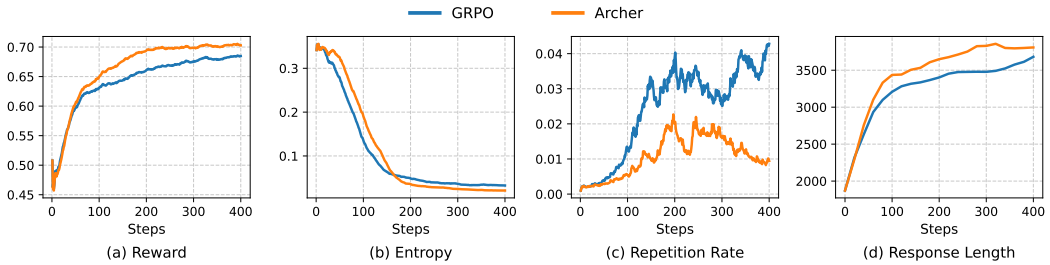


Figure 4: The training dynamics curves of all methods on Qwen3-4B, including (a) training reward, (b) model entropy, (c) repetition rate, and (d) response length. The curves are smoothed with EMA for better visualization.

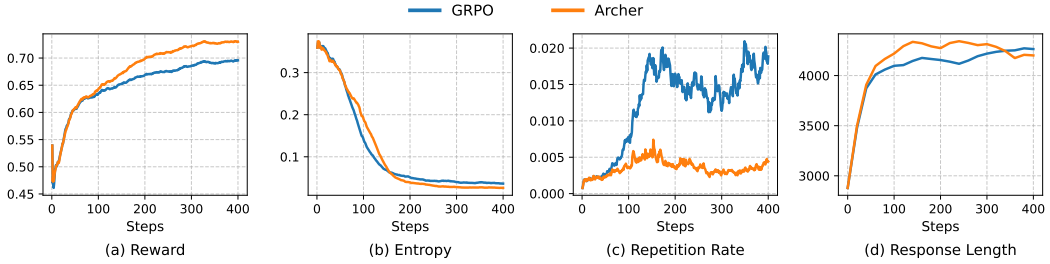


Figure 5: The training dynamics curves of all methods on Qwen3-8B, including (a) training reward, (b) model entropy, (c) repetition rate, and (d) response length. The curves are smoothed with EMA for better visualization.

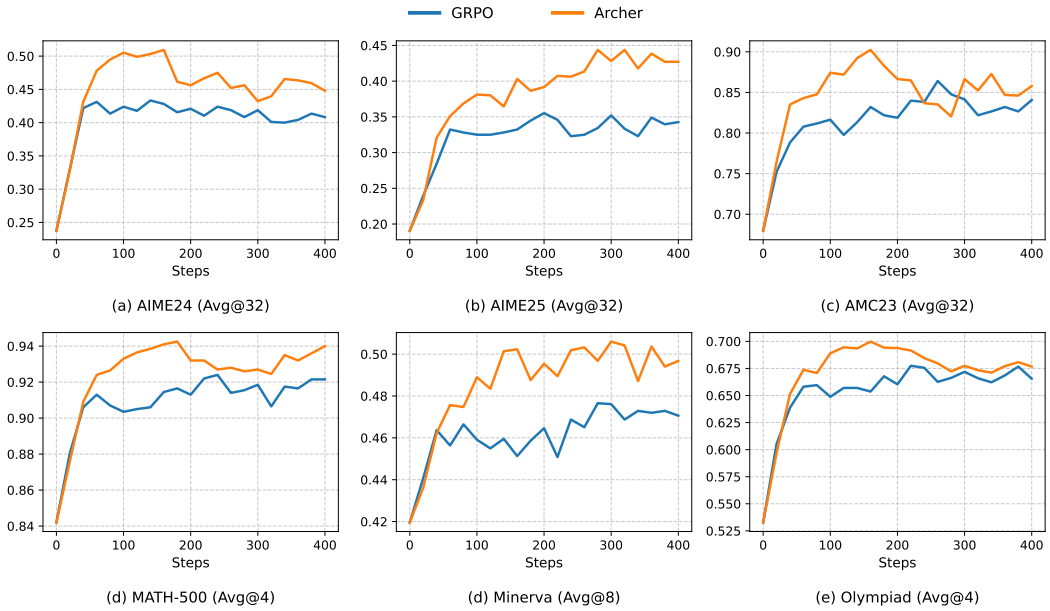


Figure 6: The test curves of all methods trained with Qwen3-4B on six mathematical benchmarks.

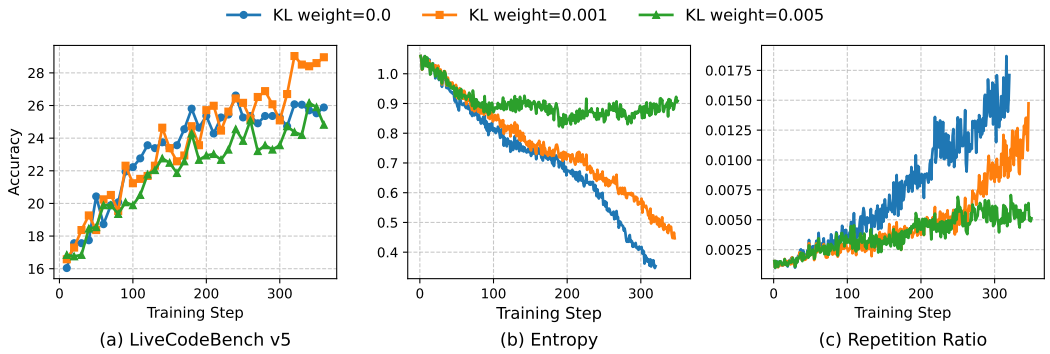


Figure 7: Effects of varying KL weights on (a) model performance on LiveCodeBench v5, (b) model entropy, and (c) repetition ratio.

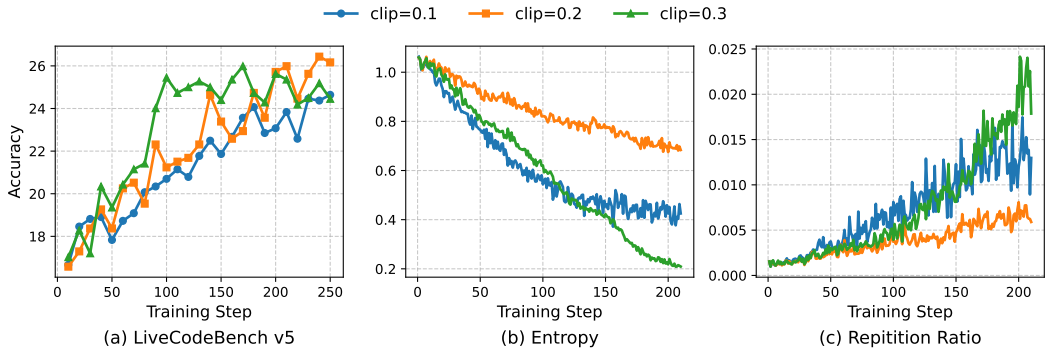


Figure 8: Effects of varying the clip threshold of low-entropy tokens on (a) model performance on LiveCodeBench v5, (b) model entropy, and (c) repetition ratio.

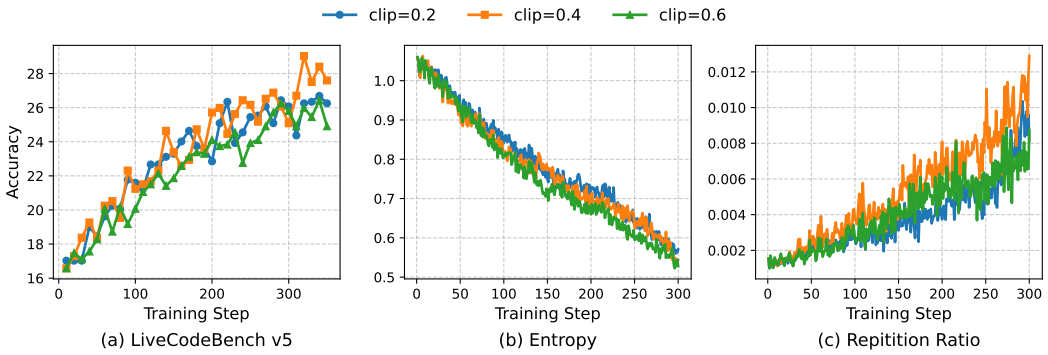


Figure 9: Effects of varying clip value on high-entropy tokens on (a) model performance on LiveCodeBench v5, (b) model entropy, and (c) repetition ratio.

B.3 Impact of Clip Ranges on High-Entropy Tokens

B.4 Mutual Enhancement Between Math RL and Code RL

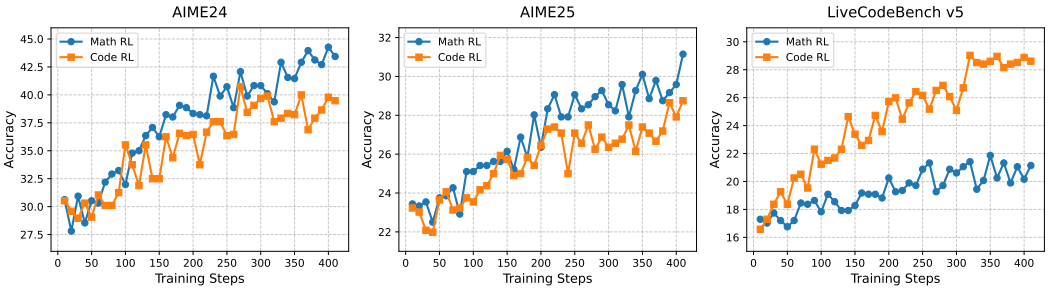


Figure 10: Model performance on AIME24, AIME25, and LiveCodeBench v5 of math RL and code RL.

Figure 10 shows results on AIME24, AIME25, and LiveCodeBench v5, comparing RL applied to math tasks (math RL) and code tasks (code RL). We observe that RL training in either domain leads to significant performance improvements not only in-domain but also on out-of-domain (OOD) benchmarks.

To analyze the source of these cross-domain improvements, we evaluate the base model and its math/code RL variants on OOD benchmarks (LiveCodeBench v5 and AIME24/25), measuring problem-level accuracy across all tasks. Unlike AceReason-Nemotron [3], which attributes the benefits of math RL on code tasks primarily to the presence of math-related subdomains (e.g., Algebra, Counting, Combinatorics), our results suggest a different explanation: performance improvements correlate more strongly with the intrinsic difficulty of the problems rather than their topical categories. Specifically, problems where the base model already achieves relatively high accuracy tend to benefit most from RL training, as shown in Figure 11 and Figure 12.

A closer analysis of the problems with notable improvement in Figure 11 shows that RL training **does not introduce fundamentally new knowledge** beyond what is already present in the base model's outputs. This observation applies to both less challenging problems (where the base model already performs well) and more challenging ones. Instead, the improvements **mainly result from enhanced reasoning capabilities**. We identify three main areas of improvement:

- **Enhanced Structural Organization**: Responses demonstrate a clearer logical flow and improved structural coherence.
- **Increased Attention to Details**: Models are more careful with edge cases and boundary conditions. This effect is especially clear in the Code-RL model, likely because boundary handling is important in programming tasks.

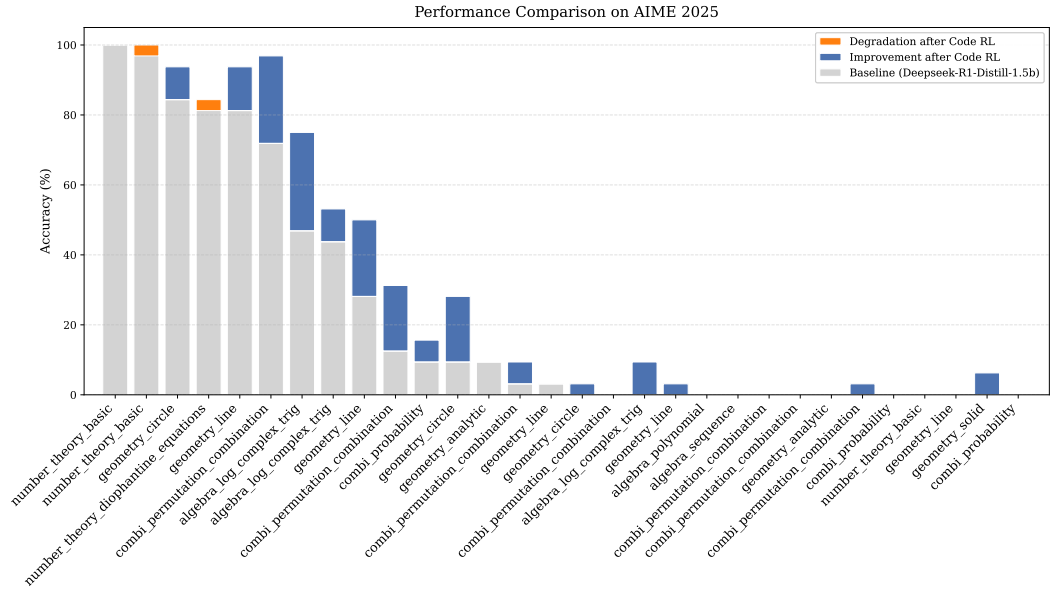


Figure 11: Problem-level accuracy comparison between the base model and RL-trained model.

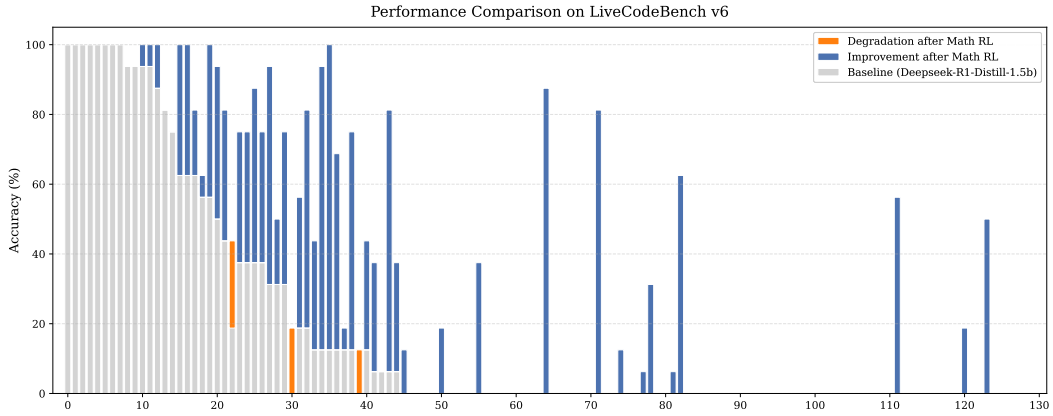


Figure 12: Problem-level accuracy comparison on LiveCodeBench v6 between the base model and Math RL trained model.

- **Improved Contextual Consistency**: RL-trained models are more accurate at integrating and summarizing previous reasoning steps. In contrast, the base model sometimes produces final answers based on incorrect intermediate reasoning even if some steps are correct, which leads to inconsistencies.

These findings further support our main claim: the main way RL improves model capability **is not by changing stored knowledge or basic skills** (such as arithmetic), **but by better integrating and optimizing existing abilities** through structured logical behavior such as reflection and planning. At the same time, this provides empirical support for the effectiveness of our proposed dual-token constraint training strategy.