Prune as You Generate: Online Rollout Pruning for Faster and Better RLVR

Anonymous ACL submission

Abstract

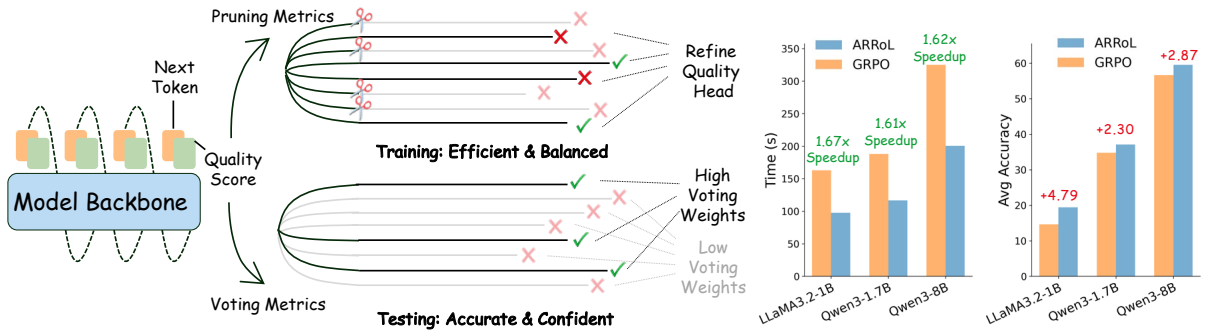
Reinforcement Learning with Verifiable Rewards (RLVR) has significantly advanced the reasoning capabilities of Large Language Models (LLMs). However, methods such as GRPO and DAPO suffer from substantial computational cost, since they rely on sampling many rollouts for each prompt. Moreover, in RLVR the relative advantage is often sparse: many samples become nearly all-correct or all-incorrect, yielding low within-group reward variance and thus weak learning signals. In this paper, we introduce ARRoL (**A**ccelerating **R**LV**R** via **o**nline Ro**L**lout Pruning), an online rollout pruning method that prunes rollouts during generation while explicitly steering the surviving ones more correctness-balanced to enhance learning signals. Specifically, ARRoL trains a lightweight quality head on-the-fly to predict the success probability of partial rollouts and uses it to make early pruning decisions. The learned quality head can further weigh candidates to improve inference accuracy during test-time scaling. To improve efficiency, we present a system design that prunes rollouts inside the inference engine and re-batches the remaining ones for log-probability computation and policy updates. Across GRPO and DAPO on Qwen-3 and LLaMA-3.2 models (1B-8B), ARRoL improves average accuracy by $+2.30$ to $+2.99$ while achieving up to $1.7\times$ training speedup, and yielding up to $+8.33$ additional gains in average accuracy in test-time scaling.

1 Introduction

Reasoning abilities of Large Language Models (LLMs) have recently gained great success in many domains such mathematical problem solving and code generation (Jaech et al., 2024; Guo et al., 2025). Reinforcement Learning with Verifiable Rewards (RLVR) (Lambert et al., 2024; Yu et al., 2025) as a critical technique plays an important role in enhancing reasoning ability of LLMs. A representative method is Group Relative Policy Optimization (GRPO) (Shao et al., 2024), which utilizes binary rewards such as correctness of a logical problem as learning signals and compute advantages within a group of rollouts per prompt. However, such methods are largely constrained by high computational cost (Xu et al., 2025; Lin et al., 2025). During training, each prompt requires generating a large group of rollouts, which is computationally expensive, making training expensive and limiting the practicality of RLVR at scale.

To mitigate training cost, prior work has explored several directions. Some studies (Lin et al., 2025; Xu et al., 2025) reduce the number of rollouts used for gradient estimation and policy updates. However, these methods typically manipulate rollouts at the post-generation level; therefore, they do not reduce rollout generation time, which can limit end-to-end speedups. Other studies employ speculative decoding to accelerate rollout generation (He et al., 2025; Liu et al., 2025), but they rely on historical sequences from previous epochs, which may vary and not suitable for commonly adopted small epochs settings. Moreover, they do not explicitly address a key issue in RLVR with binary rewards (0/1): when rewards within a group are highly imbalanced (e.g., mostly correct or mostly incorrect), the within-group reward diversity becomes low, leading to weak learning signals (Bae et al., 2025). In the extreme case where a group collapses to all 0s or all 1s, the group-normalized advantages can degenerate to zero, resulting in a vanishing policy gradient. This motivates a key question: *Can we reduce rollout cost while strengthening learning signals?*

To answer this question, we propose an online rollout pruning method that *carefully selects a correctness-balanced subset of rollouts for training during rollout generation* using a quality predictor. Concretely, we train a lightweight model head to score early-stage partial rollouts and map the score to an estimated success probability. The rollouts



(a) Illustration of ARRoL Method (b) Time Comparison (c) Accuracy Comparison

Figure 1: **ARRoL overview and results.** (a) ARRoL uses a quality head to score partial rollouts, enabling early pruning for efficient and reward-balanced training, and the scores can also be used as voting weights for test-time scaling. (b) Wall-clock time comparison between ARRoL and GRPO across different model backbones, showing consistent speedups. (c) Accuracy comparison, where ARRoL improves average accuracy over GRPO.

with final rewards can naturally be used as training data of rollouts, and it introduces negligible overhead.

We compare the quality scores with other heuristic metrics, such as DeepConf (Fu et al., 2025), and show that the scores generated by a learnable head are better than trace confidence across datasets, as the latter can be biased by patterns (e.g., reflections or formula-rich text) and may misalign with final correctness. Then, we prune the early-stage rollouts based on the quality scores to make the correctness of remaining rollouts more balanced. This yields a "less is more" effect: fewer rollouts continue to generate and involve in advantage computation, leading to less computation cost, while the remaining rollouts provide stronger learning signals due to improved balance. Furthermore, the learned quality head can naturally serve as a correctness predictor to weight candidates in *test-time scaling* to improve accuracy instead of naive majority vote.

To implement online pruning during generation and improve efficiency, we integrate pruning into a standard frontend–backend RL training architecture. The backend evaluates rollouts by the quality head at an early stage and immediately removes pruned sequences from the request pool, allowing the scheduler to reallocate freed capacity to other active sequences and reduce overall generation time. The frontend receives pruning masks, filters pruned rollouts, and re-batches the survivors for log-probability computation and optimization, leading to less computational cost as well.

In summary, our key contributions are as follows:

- **Online Rollout Pruning.** We propose an online, quality-head-guided rollout pruning strategy, ARRoL, which explicitly controls within-group reward balance while reducing compute, improving average accuracy of GRPO/DAPO training on Qwen-3 and LLaMA-3.2 models (1B-8B) by $+2.30$ to $+2.99$.

- **Test-time Scaling.** We leverage the trained quality head at inference time as voting weights for test-time scaling, improving final-answer aggregation, leading to $+8.33$ gains in average accuracy.

- **System Speedup.** We present a system design that realizes end-to-end speedups by pruning inside the generation backend and re-batching survivors in the frontend, achieving $1.6-1.7\times$ speedup.

2 Related Work

Efficient RLVR. Recent studies improve the efficiency of RLVR from different angles. Some modify rollout construction. For example, S-GRPO (Dai et al., 2025) derives a serial group of rollouts from a single trajectory, which may reduce trajectory diversity. Others leverage historical information. GRESO (Zheng et al., 2025) skips likely uninformative prompts before rollouts. RhymeRL (He et al., 2025) reuses historical rollout tokens by speculative decoding. However, these speedups rely on historical information and can be limited in cold-start settings. Another line targets post-rollout optimization. CPPO (Lin et al., 2025) prunes generated completions, and

PODS (Xu et al., 2025) downsamples a large rollout pool, to accelerate the update phase. However, these methods do not directly reduce (and can even increase) token-generation cost during rollout. Several works target to accelerating rollout generation. Spec-RL (Liu et al., 2025) and FastGRPO (Zhang et al., 2025) employ speculative decoding, while FlashRL (Yao et al.) and QeRL (Huang et al., 2025) use low-precision/quantized rollouts to speed up token generation. In contrast, our method uses a lightweight logits-based probe to predict rollout utility online, enabling training-time pruning with controlled signal quality and a unified criterion that also supports test-time rollout filtering.

Test-time Scaling. Test-time scaling (TTS) improves reasoning performance by allocating additional compute at inference time without modifying model parameters. TTS is commonly categorized into sequential and parallel strategies. Sequential TTS increases compute along a single trajectory by extending reasoning process or revisiting the initial answer. For instance, s1 (Muennighoff et al., 2025) proposes budget forcing to terminate trajectories early or append double-check tokens to encourage rethinking. Other work studies the underthinking phenomenon and triggers deeper deliberation if necessary (Wang et al., 2025). In contrast, parallel TTS samples multiple trajectory candidates and aggregates them, including Self-Consistency (Wang et al., 2022), Best-of-N (Zhou et al., 2022), and adaptive voting (Snell et al., 2024). Recent studies further explore confidence-based TTS. DeepConf (Fu et al., 2025) uses log-probability-based confidence to prune low-confidence trajectories, while CGES (Aghazadeh et al., 2025) employs heuristic estimates or reward models to early-stop the sampling process. However, these heuristic confidence signals are typically not guaranteed to align with final-answer correctness, so their reliability may degrade under distribution shift or in out-of-domain settings.

3 Preliminaries

Trace Confidence. Recent work leverages model-internal uncertainty signals to evaluate the quality of an LLM-generated trace. Given the predicted token distribution $P_t(\cdot)$ at position t, token confidence is defined as $H_t = -\sum_{j=1}^{V} \log P_t(j)$, where V is the vocabulary size. Self-uncertainty (Kang et al., 2025) defines trace confidence as the average token confidence over the trace. DeepConf (Fu et al., 2025) further improves effectiveness and efficiency by computing window-level confidence. Specifically, it averages token uncertainty within a fixed-size sliding window w: $H_w = \frac{1}{|w|}\sum_{t\in w} H_t$, and then aggregates $\{H_w\}$ along the trace, e.g., using the minimum window value or the average of the bottom 10% windows as the trace-level score.

Reinforcement Learning with Verifiable Rewards (RLVR). Let the large language model be the policy π_θ that, given a prompt x, generates a rollout o that contains the reasoning trace and the final answer y. Assume a dataset $\mathcal{D} = \{(x_i, a_i)\}_{i=1}^{N}$, where a_i is the ground-truth answer to the prompt x_i. We define a verifiable reward $R(y_i, a_i) = \mathbb{1}[y_i \equiv a_i]$, where $\mathbb{1}[\cdot] \in \{0, 1\}$ is an indicator that evaluates whether y_i and a_i are equivalent, e.g., mathematically equivalent.

Group-Relative Policy Optimization (GRPO). GRPO (Shao et al., 2024) estimates advantages using the relative performance within a group of answers for the same prompt, without a value model. The objective is:

$$J(\theta) = \mathbb{E}_{(x,a)\sim\mathcal{D},\{o_i\}_{i=1}^{G}} \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\min[r_{i,t}A_i, \text{clip}(r_{i,t}, 1-\epsilon, 1+\epsilon)A_i] - \beta\cdot \text{KL}(\pi_\theta||\pi_{ref}),$$

where G is the group size, $r_{i,t} = \frac{\pi_\theta(o_{i,t}|o_{i,<t})}{\pi_{\text{ref}}(o_{i,t}|o_{i,<t})}$ is the importance ratio, and $A_i = \frac{R(y_i,a_i)-\text{mean}(\{R(y_j,a_j)\}_{j=1}^{G})}{\text{std}(\{R(y_j,a_j)\}_{j=1}^{G})}$ is the group-relative advantage. However, GRPO incurs substantial time cost because it must generate many rollouts per prompt and process them for log-probability computation and policy updates. Also, it can encounter sparse signal issue that rewards within a group are all 0/1 and group-normalized advantages become zero, leading to vanishing gradient.

4 Method

We introduce ARROL, a rollout pruning method to improve the efficiency of GRPO by balancing 0/1 rewards within each group. ARROL trains a lightweight model head, named quality head, to score partial rollouts, and uses these scores to select a balanced subset for training. The learned scores can also be reused as voting weights at test time. We further present a system design that realizes the wall-clock speedup in practice.

4.1 Pruning Improves Sample Balance

GRPO suffers from sparse signals when the rewards are nearly all 0s or all 1s. Intuitively, a more balanced sample group will introduce larger variance within a group, leading to non-vanishing gradients. Also, if samples are balanced, we can avoid circumstances in which some groups are dominated by a few minority samples, leading to sparse and noisy learning signals. Recent studies (Bae et al., 2025) provide theoretical support that, under binary (0/1) rewards, the learning signal is proportional to the reward variance and is maximized when the pass ratio (i.e., the fraction of positive samples within a group) is close to 0.5, thereby improving the effectiveness of RL training. Based on this, letting the positive-sample ratio be ρ, we further show that rollout pruning can push the empirical ratio toward ρ (ideally $\rho = 0.5$), improving sample balance beyond its efficiency gains.

Lemma 4.1 (Existence of a Corrective Pruning)**.** *Consider a mini-batch of size G, each with a label $y_i \in \{0, 1\}$. We assume a fixed positive ratio $\rho \in [0, 1]$ and conditional independence given latent Bernoulli parameters: $Y_i|q_i^\star \sim$ Bernoulli$(q_i^\star)$. We define the batch mean $\mu^\star := \frac{1}{G}\sum_{i=1}^{G} q_i^\star$ and the pruned mean $\mu^\star_{-j} := \frac{1}{G-1}\sum_{i\neq j} q_i^\star$. If $\mu^\star > \rho$ and there exists an index j such that $q_j^\star > \mu^\star$, then pruning j strictly reduces the deviation to ρ:*

$$\left|\mu^\star_{-j} - \rho\right| < \left|\mu^\star - \rho\right|.$$

Symmetrically, if $\mu^\star < \rho$ and there exists j such that $q_j^\star < \mu^\star$, then the same conclusion holds.

Theorem 4.2 (High-probability closeness to target ρ)**.** *Under the setting of Lemma 4.1. Assume we have posterior-mean estimates $\{q_i\}_{i=1}^{G}$ satisfying the uniform accuracy condition $|q_i - q_i^\star| \le \epsilon$, $\forall i$. We define $\mu_{-j} := \frac{1}{G-1}\sum_{i\neq j} q_i$, let $\hat{\jmath} := \arg\min_{j\in[G]} \left|\mu_{-j} - \rho\right|$, $\hat{p}_{-\hat{\jmath}} := \frac{1}{G-1}\sum_{i\neq\hat{\jmath}} Y_i$, and fix any $\delta \in (0, 1)$. Then, with probability at least $1 - \delta$, we have*

$$\left|\hat{p}_{-\hat{\jmath}} - \rho\right| \le \min_{j\in[G]} \left|\mu^\star_{-j} - \rho\right| + 2\epsilon + \sqrt{\frac{\log(2/\delta)}{2(G-1)}}.$$

The proof can be found in Appendix A.1. It implies that pruning can reduce the posterior-mean deviation to a target ratio ρ, and if the posterior estimates q_i are accurate, then posterior-guided pruning is $O(\epsilon)$-close to the true-posterior pruning in terms of deviation from ρ. Therefore, setting $\rho = 0.5$ can enhance the learning signals and improve the effectiveness of training.

4.2 Quality Prediction Head

We have shown that if we have an accurate posterior estimate q_i for each rollout, we can prune rollouts to control the within-group positive ratio and improve the learning signal. However, q_i is not directly observable during generation, especially when we want to prune a rollout early. To operationalize Sec. 4.1, we first construct an early-stage *quality score* s_i for a partial rollout, and then map this score to a posterior estimate $q_i \in [0, 1]$.

Quality Score Prediction. Previous studies (Kang et al., 2025; Fu et al., 2025) introduce internal uncertainty signals, trace confidence, based on the log-probability of next tokens, which can serve as *quality scores* s_i. However, these metrics are only indirect proxies of rollout quality. Since they are computed from token-level likelihood without task supervision, they are not guaranteed to align with the final success label. As shown in Fig. 2(a), a failure example indicates that reflection-related tokens tend to receive low confidence despite being beneficial, whereas formula-heavy tokens can receive high confidence even under incorrect reasoning. Therefore, we turn to the model's hidden representations to generate quality scores of rollouts. Like the next token prediction head in language models, we can also add a *rollout quality head* to the backbone model, which can be a simple MLP. Since RL naturally provides labeled rollouts, we can train the quality head on-the-fly using cross-entropy loss, whose gradient will be detached from backbone model to avoid possible overhead. To evaluate the effectiveness of the scores from quality head, we collect 4,000 rollouts from two datasets. As shown in Fig. 2(b), the scores given by quality head as quality scores can distinguish the correct rollouts from incorrect ones, with separable distributions of the two categories. Also, quality head scores show a stable Spearman rank correlation across datasets (Fig. 2(c)).

Detection Length. Training-time pruning requires choosing an intermediate length to evaluate the quality score. Fig. 2(d) reports the correlation between intermediate quality head scores and final correctness, as well as the generation time to reach each length. We find that early detection is reliable, and choose $L_{\text{detect}} = 512$ to balance pruning reliability and time cost.

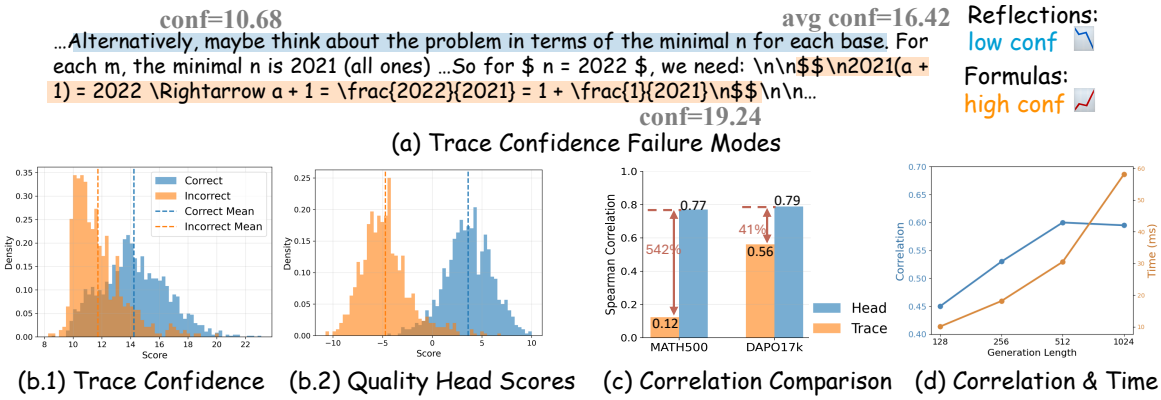


Figure 2: (a) Trace Confidence Failure Modes: Reflection-related tokens tend to receive low confidence despite being beneficial, whereas formula-heavy tokens can receive high confidence even under incorrect reasoning. (b) Distribution Comparison. Trace confidence in (b.1) is less separable between correct/incorrect than quality head scores in (b.2). (c) Correlation Comparison. Quality-head scores achieve consistently higher correlation, measured by Spearman rank correlation between the predicted scores (quality scores or trace confidence) and the binary correctness of final answers on the Math500 and Dapo17k datasets. (d) Generation Length v.s. Correlation & Time Cost. The time cost increases as the generation length increases, while the correlation plateaus when the length reaches 512. All the data is generated by Qwen3-4B model on 400 prompts from Dapo17k and Math500 dataset with 10 rollouts per sample.

Probability Calibration. Given the quality-head score s_i, we need a probability-like posterior estimate $q_i \in [0, 1]$ to instantiate Sec. 4.1. Since the raw score scale may shift during training, we adopt an *online binned probability estimator* to map scores to posteriors (Zadrozny and Elkan, 2001). Concretely, we first normalize the score to $s'_i \in [0, 1]$ and assign it to one of B uniform bins denoted as $b(s')$. For each bin, we maintain the numbers of historical positive and negative rollouts (with a sliding buffer), and estimate the posterior success probability by:

$$q(s') = P(Y = 1|b(s')) = \frac{\pi P(b(s')|Y = 1)}{\pi P(b(s')|Y = 1) + (1 - \pi)P(b(s')|Y = 0)},$$

where $\pi = P(Y = 1)$, $P(b(s')|Y = 0)$ and $P(b(s')|Y = 1)$ are estimated by historical information from previous steps maintaining with a sliding buffer.

With $q_i = P(y = 1|s_i)$ as an estimate of the rollout success probability, we assign each rollout a *survival probability* p_i. The design goal is twofold: (i) the expected keep ratio matches a target κ, and (ii) the kept rollouts have a controlled positive ratio close to ρ. We achieve this by defining p_i as a monotonic function of $(\rho - q_i)$ and normalizing it to satisfy the keep-rate constraint. Sampling rollouts according to $\{p_i\}$ allows us to prune multiple rollouts in one step while steering the within-group balance toward ρ. More details can be found in Appendix A.3.

4.3 System Design

To enable efficient RL training, we adopt a standard frontend-backend architecture and implement rollout pruning inside the generation backend (Fig. 3). In each training step, we (i) generate rollouts, (ii) compute log-probabilities/advantages, and (iii) update the policy. The frontend orchestrates data and runs log-probability computation and policy updates, while the backend provides high-throughput rollout generation. **(i) Backend.** The frontend sends rollout-generation requests to the backend. The backend maintains a request pool and dynamically batches active sequences for GPU execution. When a rollout first reaches the detection length L_{detect}, the backend evaluates its quality and samples a pruning decision according to the survival probability (Sec. 4.2). Pruned rollouts are immediately removed from the request pool, so the scheduler can reallocate the freed capacity to other active sequences, reducing the overall generation time without lowering GPU utilization. **(ii) Frontend.** The backend returns the pruning masks together with the generated rollouts. The frontend filters out pruned rollouts and re-batches the surviving ones to compute log-probabilities and advantages, followed by policy optimization. This reduces the log-probability computation and backpropagation cost roughly in proportion to the number of surviving rollouts. **(iii) Quality head.** Each rollout is

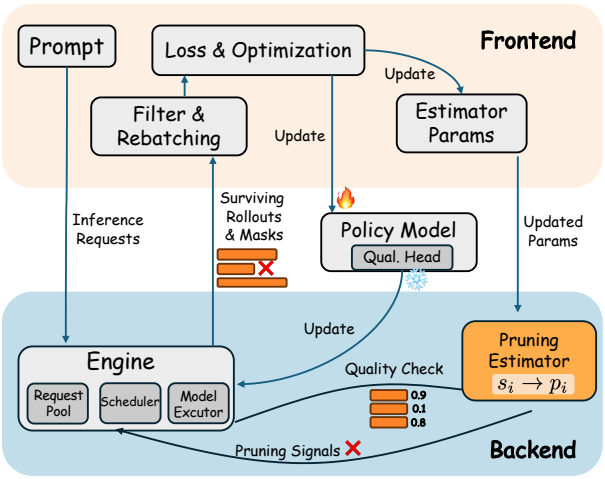


Figure 3: Illustration of System Design.

naturally labeled by its final reward, so we can collect training data for the quality head on the fly. We update the quality head with a cross-entropy loss, while stopping gradients to the backbone model. As a result, the additional overhead is negligible. For more details, please refer to Fig. 3 and Appendix A.4.

4.4 Test-time Scaling.

At test time, the trained quality head can also naturally serve as a correctness predictor. Given a set of completed reasoning traces, we apply the head to obtain a score s_i for each candidate. Instead of a naive majority vote, we use these scores to calculate voting weights. Since s_i is an uncalibrated logit-like score, we convert scores to rank-based weights by sorting candidates and linearly rescaling their ranks to $[0, 1]$.

5 Experiment

5.1 Experimental Settings

We build system based on verl framework for training and vLLM framework as inference engine. We conduct experiments on LLMs of different sizes and different series: Qwen-3 (1.7B, 4B, 8B), and LLaMA-3.2 (1.7B). We compare vanilla GRPO (Shao et al., 2024) and DAPO (Yu et al., 2025) with their ARRoL-equipped variants on Math500 (Hendrycks et al., 2021), OlympiadBench (He et al., 2024), and MinervaMath (Lewkowycz et al., 2022) using average accuracy, and on AMC'23, AIME'24, and AIME'25 using pass@16. When training with our method, we use a cold-start period of 20 steps to initialize the quality head and the pruning estimator. For test-time scaling, we report maj@32 and compare our method against vanilla GRPO and trace score from DeepConf (Fu et al., 2025). More details about the metrics and datasets can be found in Appendix A.2. Models are trained on the Dapo-Math-17K (Yu et al., 2025) dataset with a maximum sequence length of 8,192. The learning rate is set to 1×10^{-6}, and the group size is set to 16. Other hyperparameters include $\kappa = 0.5$, $\rho = 0.5$, $L_{\text{detect}} = 512$. For GRPO algorithm, we set $\epsilon_{low} = \epsilon_{high} = 0.2$, and for DAPO algorithm, we change ϵ_{high} to 0.28. The experiments are conducted on NVIDIA GH200 GPUs.

5.2 Main Results

Performances on GRPO. We report performance comparisons between vanilla GRPO and GRPO equipped with ARRoL, as shown in Table 1. Across most benchmarks, ARRoL consistently outperforms vanilla GRPO, improving average accuracy by $+2.30$ to $+2.87$ on the Qwen-3 series and by $+2.86$ on LLaMA-3.2. Notably, the gains are larger on harder benchmarks: for example, ARRoL improves AMC'23 by $+7.50$ on Qwen-3-1.7B and improves AIME'24 by $+10.00$ on Qwen-3-8B (and $+6.67$ on AIME'25). Meanwhile, we observe small regressions on a few datasets (e.g., Minervamath on Qwen-3-1.7B/4B), but the overall improvement remains consistent. In addition to better performance, ARRoL substantially reduces training cost, achieving a stable $1.6-1.7\times$ end-to-end speedup across model sizes. These results support a "less is more" effect: pruning yields fewer but more balanced samples, leading to both higher accuracy and better efficiency, and the gains are robust across model families and sizes. Finally, the quality head reaches $\sim$80% prediction accuracy (e.g., 82.37% on Qwen3-1.7B), indicating it can be reliably trained within our pipeline for early-stage pruning decisions.

Performances on DAPO. We further evaluate ARRoL on DAPO by comparing vanilla DAPO with its ARRoL-equipped variant. The results are reported in Table 2. Overall, ARRoL achieves the best performance on most benchmarks, improving average accuracy by $+2.99$ while maintaining a $1.70\times$ end-to-end speedup. These results suggest that ARRoL generalizes well across RLVR algorithms and delivers consistent efficiency gains.

Test-time Scaling. To evaluate the quality head at inference time, we compare ARRoL against vanilla majority voting and DeepConf (Fu et al.,

Table 1: Performance Comparison on GRPO. We compare our method with vanilla GRPO on six benchmarks across four models, and also report speedup.

Method	Math500	Minervamath	OlympiadBench	AMC'23	AIME'24	AIME'25	Avg	Speedup
				Qwen-3-1.7B-Base				
GRPO	60.89	**17.65**	18.55	75.00	20.00	**16.67**	34.79	-
+ARRoL	**62.30**	16.91	**20.81**	**82.50**	**23.33**	**16.67**	**37.09**	1.61×
				Qwen-3-4B-Base				
GRPO	79.64	**30.88**	31.37	87.50	36.67	40.00	51.01	-
+ARRoL	**80.04**	28.67	**33.33**	**92.50**	**40.00**	**46.67**	**53.54**	1.63×
				Qwen-3-8B-Base				
GRPO	81.25	32.36	**34.69**	**95.00**	56.67	**40.00**	56.66	-
+ARRoL	**81.45**	**34.19**	33.18	**95.00**	**66.67**	**46.67**	**59.53**	1.62×
				LLama-3.2-1B-Instruct				
GRPO	24.20	3.31	2.26	45.00	13.33	0.00	14.63	-
+ARRoL	**29.03**	**4.04**	**4.37**	**47.50**	**16.67**	**3.33**	**17.49**	1.67×

Table 2: Performance Comparison on DAPO. We compare our method with vanilla DAPO on six benchmarks on Qwen-3-1.7B-Base, and also report speedup.

Method	Math500	Minervamath	OlympiadBench	AMC'23	AIME'24	AIME'25	Avg	Speedup
				Qwen-3-1.7B-Base				
DAPO	**62.10**	**20.96**	20.51	**75.00**	20.00	20.00	36.43	-
+ARRoL	**62.10**	**20.96**	**20.97**	72.50	**33.33**	**26.67**	**39.42**	1.70×

2025), which uses log-likelihood-based trace confidence as voting weights for final-answer aggregation. As shown in Table 3, while DeepConf improves over majority vote, the learned quality head provides consistent gains across datasets and models, yielding up to $+8.33$ additional improvement over DeepConf. These results suggest that the quality head trained during RLVR can serve as reliable confidence weights for test-time voting, outperforming model-intrinsic heuristic signals (e.g., DeepConf) that are not guaranteed to align with final-answer correctness.

5.3 Ablation Studies

Comparison with Random Pruning. To validate the effectiveness of ARRoL, we compare it with random pruning in Table 4. ARRoL consistently outperforms random pruning across datasets. We further report the within-group positive ratio during training. Specifically, for each prompt group we compute $\hat{\rho}$ as the fraction of positive (reward=1) rollouts during training, and report $\mathbb{E}[\hat{\rho}]$ and $\mathbb{E}[\hat{\rho}(1-\hat{\rho})]$ (average across groups). For binary rewards, $\hat{\rho}(1-\hat{\rho})$ is proportional to the within-group reward variance and is maximized at $\hat{\rho} = 0.5$ (Bae et al., 2025), indicating stronger non-degenerate learning signals for group-normalized updates. As shown in Table 4, ARRoL drives groups closer to balanced outcomes 0.5 and increases $\mathbb{E}[\hat{\rho}(1-\hat{\rho})]$, which is consistent with stronger learning signals and better final performance.

Efficiency Decomposition. We further analyze the source of the efficiency gains by decomposing training time into different phases, as shown in Table 5. Overall, our method accelerates all phases. For log-probability computation and model updates, the time is reduced by about $2\times$, since pruning discards roughly half of the rollouts. In contrast, the speedup for rollout generation is smaller ($1.46\times$), because we first generate all sequences up to a threshold length L_{thres} before pruning half of the rollouts.

Ablation Study on keep ratio κ. In our main experiments, we set the keep ratio κ to 0.5. To study

Table 3: Performance Comparison of Test-time Voting. We compare our method with GRPO and Deepconf method on six benchmarks across three models.

Method	AMC'23	AIME'24	AIME'25
Qwen-3-1.7B-Base			
Majority	55.0	16.7	3.3
Deepconf	57.5	16.7	6.7
ARRoL	**60.0**	**23.3**	**13.3**
Qwen-3-4B-Base			
Majority	72.5	26.7	20.0
Deepconf	72.5	33.3	23.3
ARRoL	**82.5**	**36.7**	**26.7**
Qwen-3-8B-Base			
Majority	75.0	23.3	26.7
Deepconf	80.0	23.3	23.3
ARRoL	**85.0**	**33.3**	**33.3**
LLama-3.2-1B-instruct			
Majority	10.0	0.0	**0.0**
Deepconf	15.0	3.3	**0.0**
ARRoL	**17.5**	**10.0**	**0.0**

Table 4: Performance Comparison against Random Pruning. $\hat{\rho}$ is the fraction of positive (reward=1) rollouts during training. For binary rewards, $\hat{\rho}(1-\hat{\rho})$ is proportional to the within-group reward variance and is maximized at $\hat{\rho} = 0.5$.

Method	AMC23	AIME24	AIME25	$\mathbb{E}[\hat{\rho}]$	$\mathbb{E}[\hat{\rho}(1-\hat{\rho})]$
Qwen-3-4B-Base					
Random	79.34	26.47	31.98	0.32	0.21
ARRoL	80.04	28.67	33.33	0.40	0.23
LLama-3.2-1B-instruct					
Random	22.18	2.94	3.02	0.14	0.11
ARRoL	29.03	4.04	4.37	0.23	0.14

Table 5: Efficiency decomposition for Qwen-3-1.7B-Base across training phases: rollout generation, log-probability computation, and model update.

Time/s	Generation	Logprob	Update
GRPO	106.82	18.40	63.05
ARRoL	72.96 (1.46×)	10.02 (1.84×)	30.26 (2.08×)

how κ affects both effectiveness and efficiency, we evaluate several values of κ. As shown in Table 6, a smaller κ yields larger speedup since fewer rollouts are kept during pruning. Performance generally improves as κ decreases, suggesting that pruning can also help by selecting more balanced sample subset. However, when $\kappa = 0.25$, too many rollouts are removed, leading to a slight performance drop. Overall, $\kappa = 0.5$ provides a good trade-off between accuracy and efficiency.

Table 6: Average accuracy on Math500, MinervaMath, and OlympiadBench, and training speedup under different rollout keep ratios κ for Qwen-3-1.7B-Base.

κ	**0.25**	**0.5**	**0.75**	**1**
Avg Acc	32.46	33.34	32.68	32.36
Speedup	2.33×	1.61×	1.17×	1.00×

Wall-clock convergence. We evaluate wall-clock convergence by plotting training reward against wall-clock time, as shown in Fig. 4. Compared with GRPO, our pruning based training reaches the same reward level in less time and attains higher reward earlier across most of training, indicating improved time-to-reward and faster wall-clock convergence.

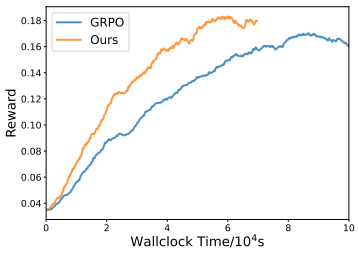


Figure 4: Wall-clock convergence of Qwen-3-1.7B-Base training.

6 Conclusion

We presented ARRoL, an online rollout pruning approach for RLVR that prunes rollouts *during* generation while explicitly steering the surviving group toward a more balanced 0/1 reward composition, strengthening learning signals. ARRoL trains a lightweight quality head on-the-fly to predict the success probability of partial rollouts, and uses them to make early pruning decisions under a target keep ratio. To realize efficiency gains, we further implemented a system design. Empirically, on different models, ARRoL consistently improves accuracy while achieving up to 1.7× training speedup. Moreover, the learned quality head can also be used at test time as voting weights, yielding additional gains over naive majority voting. Overall, ARRoL demonstrates a practical "less rollouts, more learning" paradigm for efficient and effective RLVR training and test-time scaling.

Limitations

Our study mainly focuses on mathematical RLVR tasks with verifiable rewards; while the core idea (online pruning guided by a correctness predictor and balance control) is general and could be extended to other reward-based RL scenarios, such as UI interaction or tool-use agents, we do not validate these domains in this work. In addition, training-time pruning needs to generate tokens up to an intermediate detection length to evaluate partial rollouts (we set $L_{\text{detect}} = 512$), so the rollout-generation speedup can be smaller than the savings in later phases because sequences must reach this threshold before pruning takes effect.

References

Ehsan Aghazadeh, Ahmad Ghasemi, Hedyeh Beyhaghi, and Hossein Pishro-Nik. 2025. Cges: Confidence-guided early stopping for efficient and accurate self-consistency. *arXiv preprint arXiv:2511.02603*.

Sanghwan Bae, Jiwoo Hong, Min Young Lee, Hanbyul Kim, JeongYeon Nam, and Donghyun Kwak. 2025. Online difficulty filtering for reasoning oriented reinforcement learning. *arXiv preprint arXiv:2504.03380*.

Muzhi Dai, Chenxu Yang, and Qingyi Si. 2025. S-grpo: Early exit via reinforcement learning in reasoning models. *arXiv preprint arXiv:2505.07686*.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. 2025. Deep think with confidence. *arXiv preprint arXiv:2508.15260*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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

Jingkai He, Tianjian Li, Erhu Feng, Dong Du, Qian Liu, Tao Liu, Yubin Xia, and Haibo Chen. 2025. History rhymes: Accelerating llm reinforcement learning with rhymerl. *arXiv preprint arXiv:2508.18588*.

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

Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30.

Wei Huang, Yi Ge, Shuai Yang, Yicheng Xiao, Huizi Mao, Yujun Lin, Hanrong Ye, Sifei Liu, Ka Chun Cheung, Hongxu Yin, and 1 others. 2025. Qerl: Beyond efficiency–quantization-enhanced reinforcement learning for llms. *arXiv preprint arXiv:2510.11696*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Zhewei Kang, Xuandong Zhao, and Dawn Song. 2025. Scalable best-of-n selection for large language models via self-certainty. *arXiv preprint arXiv:2502.18581*.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, and 1 others. 2024. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857.

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

Zhihang Lin, Mingbao Lin, Yuan Xie, and Rongrong Ji. 2025. Cppo: Accelerating the training of group relative policy optimization-based reasoning models. *arXiv preprint arXiv:2503.22342*.

Bingshuai Liu, Ante Wang, Zijun Min, Liang Yao, Haibo Zhang, Yang Liu, Anxiang Zeng, and Jinsong Su. 2025. Spec-rl: Accelerating on-policy reinforcement learning via speculative rollouts. *arXiv preprint arXiv:2509.23232*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20286–20332.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, and 1 others. 2025. Thoughts are all over the place: On the underthinking of o1-like llms. *arXiv preprint arXiv:2501.18585*.

Yixuan Even Xu, Yash Savani, Fei Fang, and J Zico Kolter. 2025. Not all rollouts are useful: Down-sampling rollouts in llm reinforcement learning. *arXiv preprint arXiv:2504.13818*.

Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Your efficient rl framework secretly brings you off-policy rl training, august 2025. *URL https://fengyao. notion. site/off-policy-rl*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Bianca Zadrozny and Charles Elkan. 2001. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Icml*, volume 1.

Yizhou Zhang, Ning Lv, Teng Wang, and Jisheng Dang. 2025. Fastgrpo: Accelerating policy optimization via concurrency-aware speculative decoding and online draft learning. *arXiv preprint arXiv:2509.21792*.

Haizhong Zheng, Yang Zhou, Brian R Bartoldson, Bhavya Kailkhura, Fan Lai, Jiawei Zhao, and Beidi Chen. 2025. Act only when it pays: Efficient reinforcement learning for llm reasoning via selective rollouts. *arXiv preprint arXiv:2506.02177*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and 1 others. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

A Appendix

A.1 Proof of Theorems in Sec. 4.1

Consider a mini-batch of size G, indexed by $i \in \{1, \dots, G\}$. Each sample has a binary label $Y_i \in \{0, 1\}$. We assume conditional independence given latent Bernoulli parameters:

$$Y_i \mid q_i^\star \sim \text{Bernoulli}(q_i^\star),$$

We assume $\{Y_i\}_{i=1}^G$ are independent given $\{q_i^\star\}_{i=1}^G$. Here $q_i^\star := \mathbb{P}(Y_i = 1 \mid X_i)$ is the true posterior. We have an estimated posterior q_i satisfying a uniform accuracy condition

$$|q_i - q_i^\star| \le \epsilon, \quad \forall i.$$

For any index j, define the *true expected* positive ratio after removing j:

$$\mu_{-j}^\star := \frac{1}{G-1} \sum_{i \neq j} q_i^\star.$$

Likewise define the *estimated* ratio based on $\{q_i\}$:

$$\mu_{-j} := \frac{1}{G-1} \sum_{i \neq j} q_i.$$

We consider the posterior-guided pruning rule

$$\hat{j} := \arg\min_{j \in [G]} |\mu_{-j} - \rho|,$$

where $\rho \in [0, 1]$ is a fixed ratio.

Lemma 4.1 (Existence of an improving prune) *If $\mu^\star > \rho$ and there exists an index j such that $q_j^\star > \mu^\star$, then pruning j strictly reduces the deviation to ρ:*

$$\left|\mu_{-j}^\star - \rho\right| < \left|\mu^\star - \rho\right|.$$

Symmetrically, if $\mu^\star < \rho$ and there exists j such that $q_j^\star < \mu^\star$, then the same conclusion holds.

Proof. Assume $\mu^\star > \rho$. If $q_j^\star > \mu^\star$, then

$$\mu_{-j}^\star - \mu^\star = \frac{G\mu^\star - q_j^\star}{G-1} - \mu^\star = \frac{\mu^\star - q_j^\star}{G-1} < 0,$$

so $\mu_{-j}^\star < \mu^\star$. Since $\rho < \mu^\star$, moving $\mu^\star$ downward moves it toward ρ, hence $|\mu_{-j}^\star - \rho| < |\mu^\star - \rho|$. The other case is symmetric. □

Lemma A.1 (Posterior error transfers to batch ratio)**.** *Given $|q_i - q_i^\star| \le \epsilon$, $\forall i$, for any j, we have*

$$|\mu_{-j} - \mu_{-j}^\star| \le \epsilon.$$

Proof.

$$|\mu_{-j} - \mu^\star_{-j}| = \Big|\frac{1}{G-1}\sum_{i\neq j}(q_i - q_i^\star)\Big| \le \frac{1}{G-1}\sum_{i\neq j}|q_i - q_i^\star| = \frac{1}{G-1}(G-1)\epsilon \le \epsilon.$$

□

Lemma A.2 (Near-optimality of posterior-guided pruning)**.** *Let $j^\star := \arg\min_j |\mu^\star_{-j} - \rho|$ be the best (oracle) removal index. Then given $|q_i - q_i^\star| \le \epsilon$, $\forall i$, we have*

$$|\mu^\star_{-\hat{\jmath}} - \rho| \le \min_j |\mu^\star_{-j} - \rho| + 2\epsilon = |\mu^\star_{-j^\star} - \rho| + 2\epsilon.$$

Proof. By triangle inequality and Lemma A.1,

$$|\mu^\star_{-\hat{\jmath}} - \rho| \le |\mu_{-\hat{\jmath}} - \rho| + |\mu_{-\hat{\jmath}} - \mu^\star_{-\hat{\jmath}}| \le |\mu_{-\hat{\jmath}} - \rho| + \epsilon.$$

Since $\hat{\jmath} = \arg\min_j |\mu_{-j} - \rho|$,

$$|\mu_{-\hat{\jmath}} - \rho| \le |\mu_{-j^\star} - \rho| \le |\mu^\star_{-j^\star} - \rho| + |\mu_{-j^\star} - \mu^\star_{-j^\star}| \le |\mu^\star_{-j^\star} - \rho| + \epsilon.$$

Then we have $|\mu^\star_{-\hat{\jmath}} - \rho| \le |\mu^\star_{-j^\star} - \rho| + 2\epsilon$. □

Lemma A.3 (Concentration of realized ratio around its expectation)**.** *Let $\hat{p}_{-\hat{\jmath}} := \frac{1}{G-1}\sum_{i\neq\hat{\jmath}} Y_i$, $\mu^\star_{-\hat{\jmath}} := \frac{1}{G-1}\sum_{i\neq\hat{\jmath}} q_i^\star$, where $Y_i \mid q_i^\star \sim \mathrm{Bernoulli}(q_i^\star)$ are conditionally independent. Then for any $t > 0$,*

$$P\Big(\big|\hat{p}_{-\hat{\jmath}} - \mu^\star_{-\hat{\jmath}}\big| \ge t \,\Big|\, \{q_i^\star\}\Big) \le 2\exp\big(-2(G-1)t^2\big).$$

Proof. Condition on the latent parameters $\{q_i^\star\}_{i=1}^G$. Further condition on the (possibly data-dependent) pruned index $\hat{\jmath}$. Given $\hat{\jmath} = j$, the kept labels $\{Y_i\}_{i\neq j}$ remain independent Bernoulli random variables with means $\mathbb{E}[Y_i \mid \{q_k^\star\}, \hat{\jmath} = j] = q_i^\star$ for all $i \neq j$. Define centered variables $Z_i := Y_i - q_i^\star$ for $i \neq j$. Then $\{Z_i\}_{i\neq j}$ are independent, satisfy $\mathbb{E}[Z_i \mid \{q_k^\star\}, \hat{\jmath} = j] = 0$, and are bounded. Since $Y_i \in \{0,1\}$ and $q_i^\star \in [0,1]$,

$$Z_i \in [-q_i^\star,\, 1 - q_i^\star] \subseteq [-1, 1].$$

Additionally, we have

$$\hat{p}_{-j} - \mu^\star_{-j} = \frac{1}{G-1}\sum_{i\neq j}(Y_i - q_i^\star) = \frac{1}{G-1}\sum_{i\neq j} Z_i.$$

By Hoeffding's inequality (Hoeffding, 1963), for any $t > 0$, we have

$$P\Big(\hat{p}_{-j} - \mu^\star_{-j} \ge t \,\Big|\, \{q_k^\star\}, \hat{\jmath} = j\Big) \le \exp\Big(-\frac{2(G-1)^2t^2}{\sum_{i\neq j}(1-(-1))^2}\Big) = \exp\big(-2(G-1)t^2\big).$$

Applying the same bound to $-(\hat{p}_{-j} - \mu^\star_{-j})$ and taking a union bound yields

$$P\Big(\big|\hat{p}_{-j} - \mu^\star_{-j}\big| \ge t \,\Big|\, \{q_k^\star\}, \hat{\jmath} = j\Big) \le 2\exp\big(-2(G-1)t^2\big).$$

Finally, remove the conditioning on $\hat{\jmath}$:

$$P\Big(\big|\hat{p}_{-\hat{\jmath}} - \mu^\star_{-\hat{\jmath}}\big| \ge t \,\Big|\, \{q_k^\star\}\Big) = \sum_{j=1}^G P(\hat{\jmath} = j \mid \{q_k^\star\}) \cdot P\Big(\big|\hat{p}_{-j} - \mu^\star_{-j}\big| \ge t \,\Big|\, \{q_k^\star\}, \hat{\jmath} = j\Big) \le 2\exp\big(-2(G-1)t^2\big).$$

□

Theorem 4.2 (High-probability closeness to target ρ.) *Fix $\delta \in (0,1)$. Given $|q_i - q_i^\star| \le \epsilon$, $\forall i$, with probability at least $1-\delta$, we have*

$$\big|\hat{p}_{-\hat{\jmath}} - \rho\big| \le \min_j |\mu^\star_{-j} - \rho| + 2\epsilon + \sqrt{\frac{\log(2/\delta)}{2(G-1)}}.$$

Proof. By triangle inequality, we have

$$|\hat{p}_{-\hat{\jmath}} - \rho| \le |\hat{p}_{-\hat{\jmath}} - \mu^\star_{-\hat{\jmath}}| + |\mu^\star_{-\hat{\jmath}} - \rho|.$$

Use Lemma A.3 with $t = \sqrt{\frac{\log(2/\delta)}{2(G-1)}}$ and Lemma A.2, we can obtain the theorem. □

A.2 Details of the Datasets

Dapo-Math-17k. DAPO-Math-17k (Yu et al., 2025) is a curated collection of mathematical problems paired with verifiable final answers. The problems are sourced from online math resources and manual annotations, and are transformed to require an integer final answer to facilitate easy parsing. The dataset is commonly used for RLVR-style training and evaluation on math reasoning tasks.

Math500. Math500 (Lightman et al., 2023) is a subset of the MATH (Hendrycks et al., 2021) dataset, containing 500 high-school-level problems. It covers a wide range of topics, including algebra, geometry, and precalculus, and is commonly used for comprehensive evaluation of mathematical reasoning.

Minervamath. Minervamath (Lewkowycz et al., 2022) consists of 272 problems, sourced primarily from MIT OpenCourseWare courses. It is designed to evaluate the mathematical and quantitative reasoning capabilities of LLMs.

OlympiadBench. OlympiadBench (He et al., 2024) contains 8,476 Olympiad-level math and physics problems, including problems from the Chinese college entrance exam. Each problem is accompanied by expert annotations with step-by-step reasoning.

AMC'23. AMC'23 is a dataset of 40 problems from the 2023 American Mathematics Competitions (AMC). The final answer is an integer ranging from 0 to 999.

AIME'24/AIME'25. Each dataset contains 30 challenging problems from the American Invitational Mathematics Examination (AIME). These questions require deep knowledge and techniques, especially in combinatorics and geometry. The final answer is an integer ranging from 0 to 999.

For small datasets such as AMC/AIME, repeated sampling is often used to reduce evaluation variance. Results are typically reported as avg@k, i.e., the average accuracy over k independent repeats, or pass@k, i.e., whether at least one of the k samples is correct. Maj@k is also commonly used, defined as the accuracy of the majority-vote answer among k repeats.

A.3 Details of Calibration Mapping and Survival Probability Design

As stated in Sec. 4.2, in practice, our quality head outputs an uncalibrated scalar *quality score* s_i for each rollout. We first normalize the score to $s_i \in [0, 1]$ using sigmoid function. We therefore require a calibration mapping $q(s') = P(Y = 1|s'_i)$ to convert scores into posterior estimates. We design an online binned probability calibration.

We employ an estimator with B bins. Specifically, we map a score s' to a bin index $b(s') = \min(B - 1, \lfloor Bs' \rfloor)$. We maintain two labeled buffers of historical rollouts (positive/negative outcomes) and compute histogram counts

$$\begin{aligned} c_b^+ &= \#\{k : b(s_k^+) = b\}, \\ c_b^- &= \#\{k : b(s_k^-) = b\}. \end{aligned} \tag{1}$$

To avoid the situation where $c_b^{+/-} = 0$ in some bins and reduce few-sample noise, we further utilize Laplace smoothing α, and estimate the class-conditional likelihoods:

$$P(b|Y = 1) = (c_b^+ + \alpha)/(\sum_{b' \in [B]} c_{b'}^+ + \alpha B),$$

$$P(b|Y = 0) = (c_b^- + \alpha)/(\sum_{b' \in [B]} c_{b'}^- + \alpha B).$$

Given a prior $\pi = P(Y = 1)$ (estimated from the buffers), we compute posterior-mean estimates via Bayes' rule:

$$\begin{aligned} q(s) &= P(Y = 1 \mid b(s)) \\ &= \frac{\pi P(b(s) \mid Y = 1)}{\pi P(b(s) \mid Y = 1) + (1 - \pi) P(b(s) \mid Y = 0)}. \end{aligned}$$

Based on $q_i = P(Y = 1|s_i)$, we assign each rollout a survival probability using an affine function of the deviation from ρ:

$$p_i = \text{clip}(\kappa + \delta + \lambda(\rho - q_i), p_{\min}, p_{\max}), \tag{2}$$

where κ is the target keep rate, λ controls the strength of balance correction, and δ is a scalar bias. The design goal is: (i) the expected keep ratio matches a target κ, and (ii) the kept rollouts have a controlled positive ratio close to ρ. We solve for δ by binary search such that the expected keep rate matches κ under the current buffer distribution: $\mathbb{E}[p_i] = \kappa$. Finally, clipping to $[p_{\min}, p_{\max}]$ prevents overly aggressive pruning and ensures every rollout retains a non-zero chance to survive, preserving exploration diversity.

In practice, we use $\lambda = 0.5$, $B = 128$.

A.4 Algorithm Framework

Here we present an algorithm framework to better illustrate our system containing fronend and backend, as shown Algorithm 1.

A.5 Potential Risks

We only use public data, and we do not expect any personally identifying information. Our method reduces the cost of RL training, which could lower the barrier to scaling RL-based post-training and be misapplied outside math. We recommend standard safety policies and dataset hygiene when transferring to other domains.

Algorithm 1 ARROL System

Require: Dataset $\{(x_i, a_i)\}_i$, batch size B, group size G

FRONTEND

for training step $t = 1, 2, \ldots$ **do**
  Sample prompts $\{(x_i, a_i)\}_{i=1}^{B}$ and form $B \times G$ rollout requests $\mathcal{P}$
  Compute histogram params h_t (binned estimator in Eq. (1), Eq. (2))
  Stream current policy + quality-head weights to backend
  Send $(\mathcal{P}, h_t)$ to backend and wait for responses $\mathcal{R}$ (with prune flags)
  Filter pruned rollouts $\mathcal{R} \to \mathcal{R}'$, rebatch
  Compute rewards, log-probs on $\mathcal{R}'$; main RL loss + quality-head loss
  Backprop and optimizer step on policy and quality head
end for

BACKEND

while requests arrive **do**
  Receive (weights stream, $\mathcal{P}, h$); load/attach weights
  Insert $\mathcal{P}$ into request pool
  while request pool not empty **do**
    Adaptively pick a micro-batch $\{r_i\}_{i=1}^{B'}$ from the pool
    Do one forward step (prefill/decoding) to get next tokens and quality logits
    for each r_i in the micro-batch **do**
      if r_i hits L_{detect} for the first time **then**
        Compute count s_i and survival prob p_i (Eq. (1), Eq. (2))
        Prune r_i w.p. $1 - p_i$; record prune flag
      end if
      Remove completed/pruned requests from pool; mark prune flags
    end for
  end while
  Return all responses (with prune flags) to frontend
end while

A.6 Declaration of AI Assistance

We used AI tools solely for language polishing. These tools did not contribute to the experiments, analysis, results, or scientific claims, and no paragraphs were generated by AI.