REAPO: RETRY-AWARE POLICY OPTIMIZATION FOR EFFICIENT EXPLORATION IN LLM REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

RLVR improves language-model reasoning, but commonly relies on independent stochastic rollouts for exploration, which can repeatedly sample similar failed strategies. Prior work has seen success with approaches that guide exploration by introducing dependence into the sampling process by including incorrect solutions in the context for subsequent attempts. Still, these methods introduce the additional question of how to assign credit to each attempt. We propose to view the credit assignment through the lens of *Stochastic Shortest Paths*, a framework in classical reinforcement learning that seeks to minimize the expected cumulative cost to reach a terminal state (in our case, a verified successful response). We show that, under suitable simplifying assumptions, this perspective naturally leads to a simple modification to GRPO, which we call **Retry-Aware Policy Optimization (ReAPO)**. ReAPO allocates additional generation to failed trajectories, allowing successful retries to be reinforced without rewarding preceding failures, and assigns credit in a theoretically principled way to each attempt. Across Qwen3 models from 0.6B to 4B and eight mathematical and scientific reasoning benchmarks, ReAPO improves first-attempt performance over GRPO as well as natural recent baseline algorithms, while additional ablations demonstrate the utility of both serial sampling and ReAPO's credit assignment. ReAPO also outperforms GRPO with larger group size, with gains preserved under token- and wall-clock-normalized comparisons. Taken together, our results suggest that ReAPO is a simple yet effective approach for improving LM reasoning performance through improved exploration and retry-aware credit assignment.

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) has emerged as an effective approach for improving the reasoning capabilities of large language models (LLMs) without requiring human-written reasoning traces (Shao et al., 2024; Guo et al., 2025; Wen et al., 2025; Lambert et al., 2025; Hu et al., 2025; Zeng et al., 2025). Given a problem with an easily verifiable answer (e.g., formalized mathematics or compilable code), RLVR samples multiple responses from a policy, evaluates each response, and reinforces responses that receive higher rewards. Most contemporary approaches to RLVR are based on the seminal algorithm Group Relative Policy Optimization (GRPO), which estimates the quality of individual responses by comparing verified rewards within the group of sampled responses. Despite the success of GRPO, the algorithm is limited by its reliance on *policy stochasticity* to explore semantically distinct reasoning trajectories. Indeed, like many policy gradient methods, GRPO does not explicitly encourage exploration, the process by which a model actively seeks out novel reasoning strategies; moreover, this deficiency has been noted in prior work that questions the ability of RL methods to drive genuine reasoning improvements beyond the capabilities of the base model (Yue et al., 2025; Karan & Du, 2025; Huang et al., 2025b; Chen et al., 2026; Wen et al., 2025; Huang et al., 2025a).

Critically, GRPO relies on independently sampled rollouts to explore different reasoning trajectories, which can result in repeated exploration of the same incorrect paths and inefficient use of computational resources. Indeed, if a model is unlikely to solve a given problem, it may well be the case that independent rollouts make the same mistake again and again, leading to no useful training signal for the model to learn from its mistakes and wasting valuable computational resources in generating redundant and uninformative tokens. Recent work has demonstrated that introducing *dependence*

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

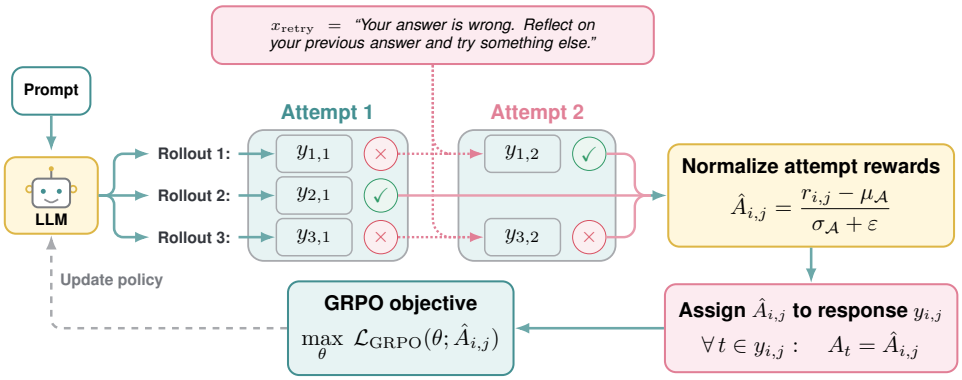


Figure 1: **Overview of ReAPO.** Each rollout begins with an independent response; failed responses receive a verifier-conditioned retry, while successful rollouts terminate. Advantages are computed for each attempt separately, ensuring that credit is assigned appropriately to both initial and retry attempts, before applying a policy update with the GRPO loss.

into the generation process can help improve coverage and potentially improve RLVR (Snell et al., 2024; Madaan et al., 2023; Sprague et al., 2026; Kumar et al., 2024; Li et al., 2026), but the best way to do so remains unclear. We thus ask: *How can we introduce structured dependence in the sampling process to improve exploration, and how should we assign credit to each attempt?*

In this work, we propose **Retry-Aware Policy Optimization (ReAPO)** (Algorithm 1), a simple yet effective modification to GRPO that explicitly encourages exploration by allowing the model to generate retries after failed attempts and assigning credit to each attempt in a theoretically principled manner. ReAPO is compatible with any verifier-based reward function and does not require additional supervision or auxiliary objectives; moreover, ReAPO can be easily integrated into existing training pipelines, requiring minimal changes to existing code.

In more detail, our motivating question breaks into two parts whose combination yields ReAPO: (i) introducing beneficial dependence into generation and (ii) correctly assigning credit to individual attempts. Prior work has demonstrated the efficacy of keeping information *in context* to encourage language models to reflect on prior mistakes and correct themselves (Kumar et al., 2024; He et al., 2026; Bensal et al., 2025; Guo et al., 2025). We build on this idea and propose *failure-conditioned repeated attempts*: given a prompt and response, we grade the answer and, if the answer is incorrect, append an instruction to the model to try again. This process can then be repeated either until a correct answer is generated or until some maximum budget of attempts (or tokens) is reached. Critically, unlike GRPO, where each attempt is fully independent, future attempts can learn from prior incorrect solutions, thereby achieving better coverage than mere independent sampling (Snell et al., 2024). While taking this idea to the extreme would entirely replace the parallel sampling of GRPO with serial sampling, in practice the system's benefits of parallelization lead us to instead rollout multiple sequences of attempts independently, thereby introducing dependence within but not among distinct rollouts. Failure-conditioning has additional benefits beyond exploration: serial sampling can be viewed as a way to dynamically assign additional compute (measured in generated tokens) to more difficult problems where early attempts are less likely to achieve success. Differentially allocating compute based on problem difficulty has been observed to lead to significant improvements in token and wall-clock efficiency (Yu et al., 2025; Xiong et al., 2025; Microsoft AI Team, 2024), and is likely partially responsible for ReAPO's efficiency.

Given a group of sequences of attempts as described above, it is not obvious how to assign credit to a given sub-attempt within a larger sequence. We propose to minimize the *expected number of incorrect attempts before a correct solution is provided*, which is an instance of a *stochastic shortest path* problem from the classical RL literature (Bertsekas & Tsitsiklis, 1991). Under natural simplifying assumptions, the resulting policy gradient computes *response-level advantages*, i.e., we treat each attempt within a sequence independently and assign each attempt's advantage only to the tokens relevant to that attempt, thereby allowing different attempts within a single sequence to receive different advantages. This approach encourages the model to produce correct solutions quickly while at the same time allowing the RL process to retain the benefits of conditioned sampling. The resulting

algorithm is represented visually in Figure 1, demonstrating how failed responses are incorporated into context and how correct and incorrect responses are scored.

We now summarize the remainder of the paper. In Section 2 we introduce the prerequisite notions to our derivation of ReAPO. Then, in Section 3, we formally derive ReAPO's objective under natural simplifying assumptions on the generation process. In Section 4, we investigate ReAPO empirically, demonstrating its improved efficacy relative to GRPO as well as several other recent baselines on several mathematical and scientific reasoning tasks at three different model scales, as well as comparisons to more competitive baselines and several ablations designed to demonstrate the utility of each intervention. We briefly discuss related work in Section A, and defer further discussion along with additional experiments and all proofs to the appendix.

2 Preliminaries

We now provide a brief overview of RLVR as well as the popular Group Relative Policy Optimization (GRPO) algorithm. As is common in RL for language models, we consider a language model to be a *policy* π_θ that maps a prompt x to a distribution over responses y. In this work, we always assume that the model is autoregressive, i.e., the response is generated one token at a time so that for each time step t, the model produces a token $y_t \sim \pi_\theta(\cdot|x, y_{<t})$ conditioned on the prompt and all previously generated tokens. To train the model, we assume that there is a *verifier* r that can determine whether a given response y is correct for a given prompt x, and our goal is to optimize the model to maximize the probability of generating correct responses. More precisely, we let $\mathcal{D} = \{x\}$ be a dataset of problems x. Given a problem x, the model generates a response $y \sim \pi_\theta(\cdot|x)$, and the verifier r produces a binary reward $r(x, y) \in \{0, 1\}$ indicating whether the response is correct, with the goal of maximizing $\mathbb{E}_{X\sim\mathcal{D}}\mathbb{E}_{y\sim\pi_{\theta(\cdot|x)}}[r(x, y)]$.

GRPO. The most popular approach to RLVR is *Group Relative Policy Optimization* (GRPO) Shao et al. (2024); Guo et al. (2025). For a given prompt x, GRPO samples a group $\{y_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}},\tau}(\cdot \mid x)$ of G responses from the current policy and obtains rewards $r_i = r(x, y_i)$ for each response. The rewards are then centered and normalized within the group to produce the response-level advantages

$$\hat{A}_i = \frac{r_i - \mu_r}{\sigma_r + \varepsilon}, \qquad \mu_r = \frac{1}{G}\sum_{i=1}^{G} r_i, \qquad \sigma_r = \sqrt{\frac{1}{G}\sum_{i=1}^{G}(r_i - \mu_r)^2}, \tag{1}$$

where $\varepsilon > 0$ provides numerical stability. GRPO minimizes the PPO loss Schulman et al. (2017)

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\left(\min\left\{\rho_{i,t}(\theta)\hat{A}_i,\ \text{clip}\left(\rho_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_i\right\} - \beta d_{\text{KL}}^{i,t}(\theta)\right)\right],$$

where $\rho_{i,t}(\theta) = \pi_\theta(y_{i,t}|x, y_{i,<t}) / \pi_{\theta_{\text{old}}}(y_{i,t}|x, y_{i,<t})$ for token $y_{i,t}$ is the importance weight, $\text{clip}(z, a, b) = \min\{\max\{z, a\}, b\}$ clips z to the interval $[a, b]$, $\varepsilon > 0$ is the clipping parameter, $\beta \geq 0$ is the KL-regularization coefficient, and $d_{\text{KL}}^{i,t}(\theta)$ is the per-token KL divergence from the reference policy. Following recent work (Yu et al., 2025; Tajwar et al., 2026; Hu et al., 2025), we set $\beta = 0$ and therefore do not apply explicit reference-policy KL regularization in our experiments; moreover we always have $\theta = \theta_{\text{old}}$.

Stochastic Shortest Paths. SSP problems are a classical subfield of RL concerned with undiscounted MDPs whose objective is to reach a terminal state while minimizing cumulative cost (Bertsekas & Tsitsiklis, 1991). In the context of LLM reasoning, we conceptualize each 'attempt' as an action and the concatenation of problem and previous attempts as a state, with the cumulative cost being the total number of attempts until a correct solution (the terminal state) is reached; for further details on classical SSPs, see Bertsekas & Tsitsiklis (1996).

3 ReAPO: Retry-Aware Policy Optimization

In this section we introduce our algorithm **Retry-Aware Policy Optimization (ReAPO)**, summarized in Algorithm 1. ReAPO has two key components on top of the standard GRPO pipeline: (1)

verifier-conditioned exploration, which allows the model to generate retries after failed attempts, and (2) *attempt-level credit assignment*, which correctly assigns credit to each attempt based on its relative performance within the group. We describe each component in detail below.

3.1 VERIFIER-CONDITIONED EXPLORATION

In GRPO, each model rollout is sampled independently and thus can repeatedly explore the same (incorrect and useless) reasoning trajectories, wasting compute and failing to learn from its mistakes. Motivated by prior work (Sprague et al., 2026; Kumar et al., 2024; Chung et al., 2025), we instead propose *verifier-conditioned exploration*. More precisely, for each problem x, ReAPO begins with the same initial group of G rollouts as standard GRPO. Each rollout first generates an independent response $y_{g,1}$. If the verifier marks this response as correct, the rollout terminates. If it is *incorrect*, however, the response is appended to the context together with a *retry prompt*, that explicitly instructs the model to reconsider its previous answer and attempt a different reasoning strategy. Our ablations demonstrate that the exact wording of the retry prompt is not critical, and we use "*Your answer is wrong. Reflect on your previous answer and try something else.*" for most of our experiments. The model then generates a second response $y_{g,2}$ conditioned on the problem, its previous response, and the verifier feedback. This process can continue until the model produces a correct response or reaches the maximum number of (per-problem) attempts K, or hits its token budget.

Because retries are generated only after failures, ReAPO dynamically expands the training group according to the observed difficulty of the problem under the current policy, which is known to help improve efficiency in modern RL for language modeling (Zhang et al., 2025; Xiong et al., 2025; Yang et al., 2026). Problems for which most initial responses are correct receive few additional generations, whereas problems with many incorrect responses receive more retries.

3.2 ASSIGNING CREDIT WITH MULTIPLE ATTEMPTS

Verifier-conditioned exploration helps solve the problem of repeated incorrect reasoning by allowing the model to condition its retries on previous failures, but it raises the question of how to properly assign credit to individual attempts within a multi-attempt trajectory. We propose minimizing the expected number of failed attempts to reach a correct solution as a natural objective. Note that in so doing, we are raising the probability of the first attempt being correct, which is the desideratum during evaluation.

More formally, fix a problem x and let y_k denote the k-th attempt generated for this problem, with $r_k = r(x, y_k)$ the binary reward; letting $\tau = \inf\{k \geq 1 \mid r_k = 1\}$ denote the index of the first successful attempt, we aim to minimize $\mathbb{E}_{\pi_\theta}[\tau - 1]$. This is an example of a *Stochastic Shortest Path* (SSP) problem, where the terminal state is a correct answer and the cost is constant per action Bertsekas & Tsitsiklis (1991). In order to derive a practical policy gradient, we make the following simplifying assumption: the rewards of attempts are sufficient statistics for the probability of a future attempt being correct, i.e., incorrect attempts affect the probability of a future attempt being correct only through the fact that they are incorrect. This is certainly an overly conservative assumption as, in practice, knowing that a prior failed attempt is wrong is likely to help the model attempt something qualitatively different, but its simplicity allows for a simple derivation. To state the next result, we introduce the following notation. Let $q_m(\theta) = \mathbb{P}_{\pi_\theta}(r_m = 1 | r_j = 0 \text{ for } j < m)$ be the probability of success on the m^{th} attempt and let $\alpha_m(\pi_\theta) = \mathbb{P}(\tau \geq m)$ be the probability that we need at least m attempts in order to be successful. Finally, let

$$c_m(\pi_\theta) = \sum_{k=m}^{\infty} \prod_{j=m+1}^{k} (1 - q_j(\theta)) \tag{2}$$

denote the expected number of additional attempts until a correct response is obtained, starting from attempt $m + 1$ given that the first m attempts have failed. We have the following result.

Proposition 1. *Fix a prompt x and let π_θ be the policy generating attempts for this problem. If τ denotes the index of the first successful attempt and $c_m(\pi_\theta)$ is defined as in equation 2, then the policy gradient of the expected number of failed attempts can be decomposed as*

$$-\nabla_\theta \mathbb{E}_{\pi_\theta}[\tau - 1] = \sum_{m=1}^{\infty} \alpha_m(\pi_\theta) c_m(\pi_\theta) \nabla q_m(\theta). \tag{3}$$

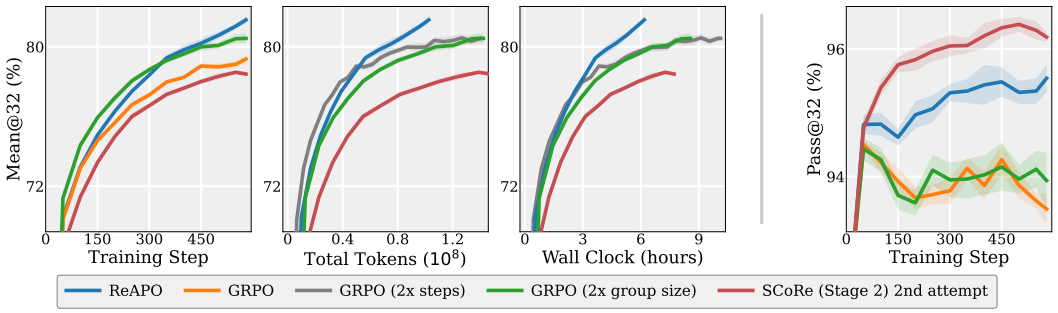


Figure 2: **Training efficiency on GSM8K-Platinum with Qwen3-0.6B-Base.** The first three panels plot accuracy (mean@32) against optimization steps, cumulative generated response tokens, and wall-clock time, respectively. The final panel plots Pass@32 against optimization steps. The GRPO-2×STEPS control extends standard GRPO to 1,160 steps in the compute-normalized panels. Curves show means across eight seeds and shaded regions show standard errors. SCoRe-Stage2 is evaluated on its second response, which is generated regardless of the first response's correctness and without verifier feedback at inference. Every other method is evaluated on its first response.

The proof of this (and all other) results can be found in Section E. Unfortunately, the decomposition in equation 3 possesses two deficiencies. First, it requires knowledge of $c_m(\pi_\theta)$, which depends on the probabilities of success for all future attempts and is therefore not directly observable or easily estimated. Second, the infinite sum may be impractical to compute in practice, necessitating truncation or approximation strategies. Thus, in order to derive a practical policy gradient estimator, we make the further assumption that $c_m(\pi_\theta)$ is *independent of* m. Under this assumption, a natural advantage estimation strategy recovers the form of equation 3 with a constant $c_m(\pi_\theta)$.

Indeed, suppose that we roll out G sequences of attempts per problem and denote by $\mathcal{A}(x) = \{r_{g,k} | g \in [G] \text{ and } k \geq 1\}$ the collection of rewards from all generated attempts. We claim that a GRPO-like advantage estimation is proportional in expectation to the true policy gradient of equation 3 under the assumption that $c_m(\pi_\theta)$ is independent of m.

Proposition 2. *Let $\mathcal{A}(x)$ be as above. Assume that $c_m(\pi_\theta) = C(\pi_\theta)$ for all m, and let*

$$\nabla_\theta \mathcal{L}(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\frac{1}{G}\sum_{g=1}^{G}\sum_{k=1}^{\tau_g} r_{g,k} \cdot \nabla_\theta \log \pi_\theta(y_{g,k} \mid h_{g,k-1})\right],$$

where $h_{g,k-1} = (x, y_{g,1}, \ldots, y_{g,k-1})$. Then it holds that $-\nabla_\theta \mathbb{E}_{\pi_\theta}[\tau - 1] = C(\pi_\theta) \cdot \nabla_\theta \mathcal{L}(\pi_\theta)$.

Under the simplifying assumptions above, weighting each generated attempt by its own unnormalized reward yields an update proportional to the negative gradient of the expected number of failed attempts. This result motivates the attempt-level credit assignment used in ReAPO. Critically, we are only applying the attempt-level advantage to the tokens making up that attempt, rather than distributing the advantage across all tokens in the trajectory (Kumar et al., 2024; Chung et al., 2025).

3.3 IMPLEMENTING REAPO

We now turn to the practical implementation of ReAPO, making two key design modifications to the method suggested by Proposition 2. First, practical constraints limit the maximum number of attempts per problem that we can actually sample, due to the quadratic increase in computational cost and linear increase in memory with the number of attempts, along with the reduced parallelization. We thus enforce a *hard cap* K on the number of attempts per problem, which acts to bias the policy gradient estimate by truncating the infinite sum in equation 3 to the first K attempts. A theoretical account of this bias can be found in Section E. Second, we follow GRPO and center and normalize the advantages by the standard deviation (offset by a small term to ensure numerical stability) (Shao et al., 2024). Exploring the effect of ablating this normalization in the style of Dr.GRPO Liu et al. (2025) is an interesting direction for future research, although we briefly discuss the theoretical consequences of normalization in Section E; centering, meanwhile, introduces a small bias proportional to $\nabla_\theta \mathbb{E}_{\pi_\theta}[1/\tau]$, which further encourages τ to be small.

Table 1: **First-attempt accuracy across model scales and benchmarks.** Qwen3-1.7B-Base results are mean ± standard error across seven seeds at the end of training; Qwen3-4B-Base results use three seeds and are evaluated at step 500.

Model / Method	Beyond AIME	AIME 2024	AIME 2025	HMMT 2025	MATH 500	Minerva	GPQA Diamond
Qwen3-1.7B-Base	1.2	4.2	3.1	0.2	57.2	21.9	10.8
GRPO	1.3 ± 0.1	11.5 ± 0.4	6.2 ± 0.3	1.3 ± 0.1	72.0 ± 0.5	30.9 ± 0.2	24.1 ± 0.7
ReAPO (ours)	$\mathbf{1.6 \pm 0.1}$	$\mathbf{13.7 \pm 0.3}$	$\mathbf{7.4 \pm 0.3}$	1.3 ± 0.2	$\mathbf{72.9 \pm 0.3}$	$\mathbf{31.7 \pm 0.2}$	$\mathbf{25.6 \pm 0.3}$
Qwen3-4B-Base	4.1	9.4	6.4	0.9	67.9	27.4	17.2
GRPO	7.6 ± 0.3	19.5 ± 1.1	16.5 ± 1.7	5.5 ± 0.8	83.0 ± 0.6	40.8 ± 0.5	28.6 ± 0.6
ReAPO (ours)	$\mathbf{8.8 \pm 0.4}$	$\mathbf{22.4 \pm 0.7}$	$\mathbf{19.0 \pm 0.9}$	$\mathbf{9.6 \pm 0.5}$	$\mathbf{85.3 \pm 0.3}$	41.3 ± 0.3	$\mathbf{30.4 \pm 0.7}$

After these modifications we are left with the following algorithm. Let $r_{g,k}$ denote the reward obtained for the k-th attempt of the g-th group. Letting $\hat{A}_{g,k}$ be defined as in equation 1, where the mean and standard deviation are computed over all $r_{g,k}$, and $|y_g| = \sum_{k=1}^{\tau_g} |y_{g,k}|$, the ReAPO objective is given by

$$\nabla_\theta \mathcal{J}_{\text{ReAPO}}(\theta) = \mathbb{E}_{x\sim\mathcal{D}} \mathbb{E}_{\pi_\theta} \left[\frac{1}{G} \sum_{g=1}^{G} \sum_{k=1}^{\tau_g} \frac{\hat{A}_{g,k}}{|y_g|} \cdot \nabla_\theta \log \pi_\theta(y_{g,k} \mid x, y_{g,<k}) \right] \tag{4}$$

As clarified in Section 2, in our experiments we always roll out on-policy (i.e. $\pi_{\theta_{\text{old}}} = \pi_\theta$) and omit the KL regularization term; both can be incorporated into equation 4 in a straightforward manner.

To reiterate, the advantage $\hat{A}_{g,k}$ is *assigned only to tokens belonging to the response* $y_{g,k}$. Thus, when an incorrect first attempt is followed by a correct retry, the failed response retains its lower advantage while the successful retry receives a higher advantage. The retry does not retroactively reward the reasoning that led to the initial failure. Likewise, a retry that repeats an incorrect answer receives another zero reward and a negative relative advantage whenever the pooled reward collection contains any successful attempts. If every generated attempt receives the same reward, all normalized advantages are zero and the problem contributes no policy-gradient signal, much as in GRPO. The pseudocode for ReAPO is presented in Algorithm 1.

4 Experiments

We now present a comprehensive evaluation of ReAPO, comparing it against standard GRPO and other relevant baselines across model scales, mathematical reasoning benchmarks, and compute budgets. We begin by describing our empirical setup (deferring additional details to Section C), then we examine the training efficiency in a number of configurations on a small model, before scaling up to larger models and more challenging benchmarks. We then thoroughly investigate the impact of the choices that underlie ReAPO's design, including the advantages of serial sampling.

4.1 Experimental Setup

Models and training data. We train Qwen3-0.6B-Base, Qwen3-1.7B-Base, and Qwen3-4B-Base Yang et al. (2025). All methods default to $G = 16$ rollouts per problem, which is the default group size used by recent works Tajwar et al. (2026), and $K = 2$ attempts in ReAPO due to the limitations of context length. The smallest model is trained on GSM-8K (Cobbe et al., 2021) with a cumulative limit of 2,048 generated response tokens per trajectory and the 1.7B/4B models on Polaris-53K An et al. (2025) with a cumulative limit of 4,096 generated response tokens per trajectory. Within each model scale, all methods use the same per-trajectory response-token limit, shared across attempts. All hyperparameters are provided in Tables 2 and 3 in the appendix.

Evaluation. We validate on standard mathematical and scientific reasoning benchmarks. For Qwen3-0.6B-Base, we evaluate on GSM8K-Platinum Vendrow et al. (2025), while for Qwen3-1.7B-Base and Qwen3-4B-Base, we evaluate on BeyondAIME ByteDance-Seed (2025), AIME

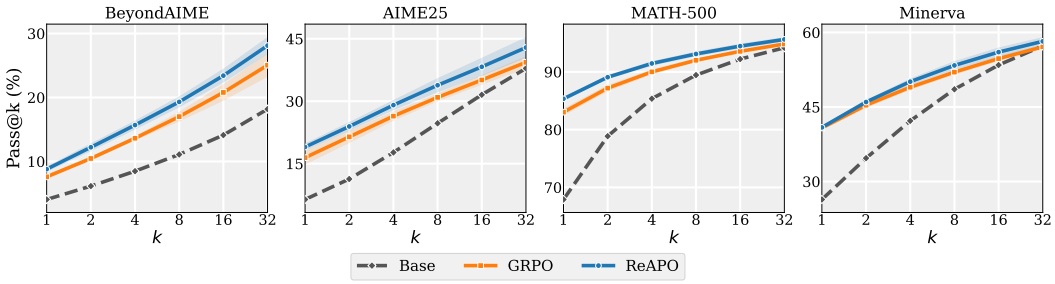


Figure 3: **First-attempt Pass@k for Qwen3-4B-Base.** Pass@k accuracy for each of BeyondAIME, AIME 2025, MATH-500, and Minerva for ReAPO, GRPO, and the base model. GRPO and ReAPO curves are means across three training seeds, with shaded regions denoting standard errors.

2024, AIME 2025 Zhang & Math-AI (2024; 2025), HMMT 2025 Dekoninck et al. (2026), MATH-500 Lightman et al. (2023); Hendrycks et al. (2021), Minerva Lewkowycz et al. (2022), and GPQA-Diamond Rein et al. (2024).

Metrics. Unless stated otherwise, we evaluate only the first response, without verifier feedback at inference. *Mean@32* is the average accuracy of 32 evaluated responses per problem, while *Pass@32* is the fraction of problems solved by at least one of those responses. These metrics use first responses for all methods except SCoRe-Stage2, for which they use second responses, which are generated regardless of whether its first response is correct. Error bars denote the standard error across seeds.

4.2 Small-Model Training Efficiency

We begin by examining the training efficiency of ReAPO relative to GRPO on the small Qwen3-0.6B-Base model. Critically, while the *maximum number of tokens* per trajectory is fixed across methods at each model scale, due to the reattempts present in ReAPO, it may well be that the *average number of tokens* per trajectory is much higher. Thus, in order to form a fair comparison among methods, we compare the training efficiency in terms of optimizer steps, the total number of generated tokens, and the wall clock time of each run. In Figure 2, we see that in each case ReAPO strictly outperforms the GRPO benchmark. Prior work has found that RL can often improve pass@1 performance without a concomitant increase in higher pass@k accuracy Yue et al. (2025); we thus also examine the effect of ReAPO on pass@32 throughout training and find directionally similar gains. Because pass@32 is almost saturated at this benchmark, however, the magnitude of the gains is much smaller.

In addition to GRPO, we consider three baselines that isolate the source of ReAPO's gains. First, we train GRPO for twice as many steps, substantially increasing both generated-token and wall-clock compute. Even with this additional compute, GRPO plateaus below ReAPO (Figure 2). Second, to test whether the gains arise simply from generating more responses rather than from serial sampling, we double the GRPO group size to $G = 32$. This matches the maximum number of responses that ReAPO can generate from $G = 16$ initial rollouts with $K = 2$, but keeps all responses independent. This baseline again plateaus below ReAPO and is substantially less token-efficient; indeed, **ReAPO requires 42% fewer tokens to reach the peak accuracy of GRPO with twice the group size and 34% fewer tokens to reach the best performance of GRPO with twice the steps**.

Third, because our comparison concerns how multi-turn training assigns rewards across attempts, we evaluate a SCoRe Stage-II-only baseline (Kumar et al., 2024)[1]. Starting from Qwen3-0.6B-Base, this baseline jointly optimizes both attempts and a progress bonus to the second-attempt reward. Its second response is generated regardless of the first response's correctness and requires no verifier feedback at inference, so we report its second-attempt performance. Our results are reported in Figure 2: due to the lack of verifier-conditioning, SCoRe is substantially less token- and compute-efficient relative to ReAPO and the different credit assignment does not encourage exploration suffi-

[1]SCoRe's Stage I is a separate RL initialization that improves second-attempt performance while KL-constraining first attempts toward the base policy, which we omit; all algorithms start from the same checkpoint.

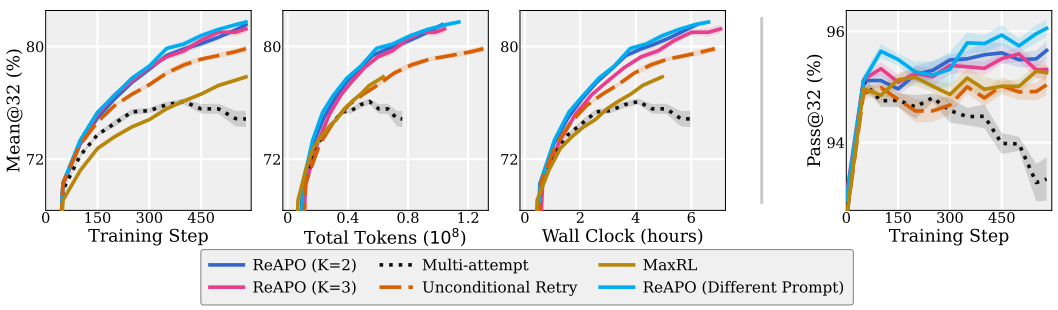


Figure 4: **Ablations on Qwen3-0.6B-Base and GSM8K-Platinum.** We compare ReAPO with $K \in \{2, 3\}$, ReAPO with an alternative retry prompt, an unconditional-retry variant that generates a second attempt regardless of whether the first attempt is correct, a multi-turn variant that assigns one advantage to the complete trajectory, and MaxRL. The first three panels report first-attempt Mean@32 against training steps, cumulative generated response tokens, and wall-clock time, respectively; the final panel reports first-attempt Pass@32 against training steps.

ciently to match ReAPO's performance in mean@32, although the pass@32 performance is slightly improved. A summary of these results can be found in Table 4 in the appendix.

4.3 Scaling Across Models and Benchmarks

While we showed above that ReAPO significantly improves training efficiency relative to GRPO on the small Qwen3-0.6B-Base model, it is important to verify whether these gains persist as we scale to larger models and more challenging benchmarks. In Table 1, we report (first-attempt) accuracy at the selected checkpoints, where at both the 1.7B and 4B scales we see that ReAPO improves on GRPO (and the base model) on nearly every benchmark. Moreover, in Figure 3, we show that ReAPO maintains its advantage over GRPO even as the inference-time sampling budget increases for the 4B scale. A similar trend is observed for the 1.7B scale, deferred to Appendix Figure 8 for the sake of space. More fine-grained results are reported in Table 5 in the appendix.

4.4 Additional Baselines and Ablations

Additional baselines. In Figure 4, we compare ReAPO to two additional baselines. MaxRL (Tajwar et al., 2026) is a recent method that aims to improve pass@k performance by modifying the advantage computation and has seen success in mathematical reasoning tasks. Multi-Attempt GRPO Chung et al. (2025) treats an entire sequence of attempts as a single multi-turn rollout. The model repeatedly attempts the problem, conditioned on its previous attempts, until it either produces a correct answer or reaches the maximum number of attempts; the entire trajectory is assigned a single reward. Consequently, failed attempts that precede a successful retry receive the same credit as the successful attempt itself. This baseline isolates the benefit of ReAPO's attempt-level credit assignment from that of verifier-conditioned multi-turn sampling alone. ReAPO consistently outperforms both baselines, demonstrating the utility of our proposed credit assignment.

Increasing the reattempt budget. One of the chief hyperparameters of ReAPO is the attempt budget K, which determines the maximum number of attempts per trajectory. While increasing K offers more opportunities to correct mistakes, it comes at the cost of more tokens per trajectory. We find that in our setup increasing K provides limited gains. We suspect that this is at least partly because each rollout is much more likely to reach its maximum generated token limit, capping the potential benefit of the third attempt. More importantly, by the end of training, more than 90% of trajectories succeed within the first two attempts, suggesting that the third attempt is rarely needed. It would be interesting to explore whether increasing the token limit or using larger models would make higher K values more beneficial, although resource constraints prevented us from doing so.

Sensitivity to prompt and verifier conditioning. Another hyperparameter of ReAPO is how we prompt the model after a failure. In Figure 4, we compare a different prompt (given in Section C.3),

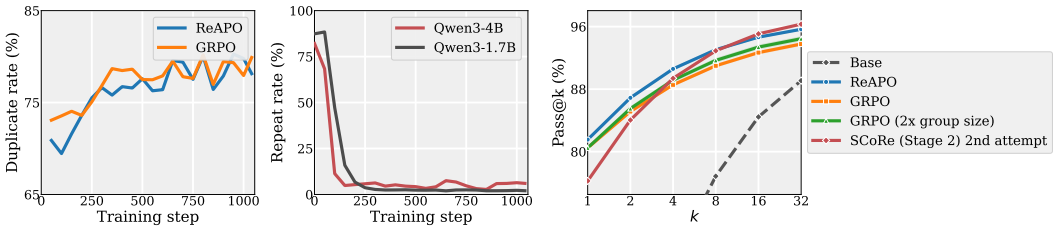


Figure 5: **Answer diversity, retry behavior, and solution coverage.** (*Left*) Percentage of incorrect first-attempt answers that are duplicates within the same question group on AIME 2025 for Qwen3-4B-Base. (*Middle*) The percentage of second-attempt answers that exactly match the failed first-attempt answer. (*Right*) Pass@k on GSM8K-Platinum for Qwen3-0.6B-Base. The SCoRe-Stage2 curve uses second responses generated regardless of first-attempt correctness; all other curves use first responses.

and find minimal difference in performance. We then consider a ReAPO variant that generates a second attempt regardless of first-attempt correctness. While the additional attempt provides more opportunities for the model to correct mistakes, because it increases the computational cost and may introduce noise from unnecessary retries, it does not perform as well as ReAPO.

4.5 Serial Sampling Encourages Exploration

We conclude our experiments by examining how serial sampling affects exploration. We first examine the extent to which retries lead to more diverse solutions relative to independent sampling. In Figure 5 (left), we compare the fraction of duplicate (independent) first attempts for ReAPO and GRPO throughout training and see that these are roughly comparable. In contradistinction to independent sampling, in the middle plot of that figure, we examine the fraction of second-attempt answers that exactly match the failed first-attempt answer, and observe that this quickly falls to near zero, demonstrating that the second attempt is highly likely to explore a different solution. A table summary of commonly generated phrase can be found in Table 7 and Table 6.

Finally, we investigate whether the increased success of ReAPO comes at the cost of solution coverage by examining how pass@k accuracy improves with k for different sampling budgets at the end of training. We find in Table 4 that ReAPO maintains or improves solution coverage relative to GRPO and matches MaxRL, which is designed specifically to improve pass@k performance.

5 Discussion and Related Work

Related work. ReAPO builds on prior work that has emphasized the importance of models reflecting on prior failed attempts during reasoning, such as self-correction, self-distillation, and multi-attempt reinforcement learning Madaan et al. (2023); Kumar et al. (2024); Guo et al. (2025); Chung et al. (2025); Bensal et al. (2025); Sprague et al. (2026). In particular, earlier works like Kumar et al. (2024) use multi-turn RL to jointly optimize multiple attempts, but does not use verifier conditioning (leading to potentially inefficient exploration) and also relies on a complicated and somewhat bespoke reward shaping. Other works like Chung et al. (2025); Bensal et al. (2025); Sprague et al. (2026) explore various aspects of multi-attempt reasoning and self-reflection, but typically do not integrate verifier-conditioned retries with attempt-level credit assignment as ReAPO does. Unlike these other methods, ReAPO's credit assignment is motivated by considering the multi-attempt RL problem as a Stochastic Shortest Paths problem (Bertsekas & Tsitsiklis, 1991) and leveraging the rich understanding of such problems from classical RL to inform the design of our algorithm. Further discussion can be found in Section A.

Limitations and future work. We demonstrated the efficacy of ReAPO across a range of mathematical and scientific reasoning tasks, but resource constraints prohibited investigation into additional domains such as code or agentic settings. We also leave further scaling of the retry budget K with larger inference-time token budgets to future work, which may well offer further gains in reasoning accuracy.

AI USE STATEMENT

The authors used generative AI to assist with drafting and editing the manuscript, writing plotting and evaluation code, and debugging. Generative AI was not used to generate datasets or experimental results. All AI-assisted content and code were reviewed and verified by the authors.

ETHICS STATEMENT

Our work uses publicly available reasoning datasets and automatically verifiable rewards, and does not involve human-subject data or personally identifiable information. Our experiments are restricted to mathematical and scientific reasoning, and we report token and wall-clock efficiency to make the computational cost of training explicit.

REPRODUCIBILITY STATEMENT

We provide anonymized code at https://anonymous.4open.science/r/ReAPO-21FC/, including training, evaluation, and plotting scripts for ReAPO and all reported baselines and ablations. Experimental settings, datasets, model configurations, hyperparameters, and additional implementation details are provided in Section C.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes, 2024. URL https://arxiv.org/abs/2306.13649.

Chenxin An, Zhihui Xie, Xiaonan Li, Lei Li, Jun Zhang, Shansan Gong, Ming Zhong, Jingjing Xu, Xipeng Qiu, Mingxuan Wang, and Lingpeng Kong. Polaris: A post-training recipe for scaling reinforcement learning on advanced reasoning models, 2025. URL https://hkunlp.github.io/blog/2025/Polaris.

Shelly Bensal, Umar Jamil, Christopher Bryant, Melisa Russak, Kiran Kamble, Dmytro Mozolevskyi, Muayad Ali, and Waseem AlShikh. Reflect, retry, reward: Self-improving llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2505.24726.

Dimitri Bertsekas and John Tsitsiklis. *Neuro-dynamic programming*, volume 5. Athena scientific Belmont, MA, 1996.

Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991. ISSN 0364765X, 15265471. URL http://www.jstor.org/stable/3690040.

Stacie Bloom, Joshua C. Brumberg, Ian Fisk, Robert J. Harrison, Robert Hull, Melur Ramasubramanian, Krystyn Van Vliet, and Jeannette Wing. Empire AI: A new model for provisioning AI and HPC for academic research in the public good. In *Practice and Experience in Advanced Research Computing (PEARC '25)*, pp. 4, Columbus, OH, USA, July 2025. ACM. doi: 10.1145/3708035.3736070. URL https://doi.org/10.1145/3708035.3736070.

Nicolas Boizard, Kevin El Haddad, Céline Hudelot, and Pierre Colombo. Towards cross-tokenizer distillation: the universal logit distillation loss for llms, 2025. URL https://arxiv.org/abs/2402.12030.

ByteDance-Seed. Beyondaime: Advancing math reasoning evaluation beyond high school olympiads. https://huggingface.co/datasets/ByteDance-Seed/BeyondAIME, 2025.

Fan Chen, Audrey Huang, Noah Golowich, Sadhika Malladi, Adam Block, Jordan Ash, Akshay Krishnamurthy, and Dylan Foster. The coverage principle: How pre-training enables post-training. In *International Conference on Learning Representations*, volume 2026, pp. 112225–112293, 2026.

Stephen Chung, Wenyu Du, and Jie Fu. Learning from failures in multi-attempt reinforcement learning, 2025. URL https://arxiv.org/abs/2503.04808.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

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, September 2025. doi: 10.1038/s41586-025-09422-z. URL https://doi.org/10.1038/s41586-025-09422-z.

Mingqian He, Yongliang Shen, Wenqi Zhang, Qiuying Peng, Jun Wang, and Weiming Lu. Star-sql: Self-taught reasoner for text-to-sql. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24365–24375, 2025.

Yinghui He, Simran Kaur, Adithya Bhaskar, Yongjin Yang, Jiarui Liu, Narutatsu Ri, Liam Fowl, Abhishek Panigrahi, Danqi Chen, and Sanjeev Arora. Self-distillation zero: Self-revision turns binary rewards into dense supervision, 2026. URL https://arxiv.org/abs/2604.12002.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

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, 2025. URL https://arxiv.org/abs/2503.24290.

Audrey Huang, Adam Block, Dylan Foster, Dhruv Rohatgi, Cyril Zhang, Max Simchowitz, Jordan Ash, and Akshay Krishnamurthy. Self-improvement in language models: The sharpening mechanism. In *International Conference on Learning Representations*, volume 2025, pp. 76687–76739, 2025a.

Audrey Huang, Adam Block, Qinghua Liu, Nan Jiang, Akshay Krishnamurthy, and Dylan J. Foster. Is best-of-n the best of them? coverage, scaling, and optimality in inference-time alignment. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025b.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation, 2026. URL https://arxiv.org/abs/2601.20802.

Muhammed Emrullah Ildiz, Halil Alperen Gozeten, Ege Onur Taga, and Samet Oymak. Learning to correct: Calibrated reinforcement learning for multi-attempt chain-of-thought. *arXiv preprint arXiv:2604.17912*, 2026.

Aayush Karan and Yilun Du. Reasoning with sampling: Your base model is smarter than you think, 2025. URL https://arxiv.org/abs/2510.14901.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning, 2024. URL https://arxiv.org/abs/2409.12917.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025. URL https://arxiv.org/abs/2411.15124.

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, 2022. URL https://arxiv.org/abs/2206.14858.

Chengpeng Li, Mingfeng Xue, Zhenru Zhang, Jiaxi Yang, Beichen Zhang, Bowen Yu, Binyuan Hui, Junyang Lin, Xiang Wang, and Dayiheng Liu. Start: Self-taught reasoner with tools. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13523–13564, 2025.

Michael Y Li, Anthony Zhan, Kanishk Gandhi, Noah D Goodman, and Emily B Fox. Quasimotto: Quasi-monte carlo test-time scaling. *arXiv preprint arXiv:2607.01179*, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

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, 2025. URL https://arxiv.org/abs/2503.20783.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

The Microsoft AI Team. Mai-thinking-1: Building a hill-climbing machine, 2024.

Benjamin Minixhofer, Ivan Vulić, and Edoardo Maria Ponti. Universal cross-tokenizer distillation via approximate likelihood matching, 2025. URL https://arxiv.org/abs/2503.20083.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

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

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning, 2026. URL https://arxiv.org/abs/2601.19897.

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, 2024. URL https://arxiv.org/abs/2409.19256.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Zayne Sprague, Jack Lu, Manya Wadhwa, Sedrick Keh, Mengye Ren, and Greg Durrett. Skillfactory: Self-distillation for learning cognitive behaviors, 2026. URL https://arxiv.org/abs/2512.04072.

Fahim Tajwar, Guanning Zeng, Yueer Zhou, Yuda Song, Daman Arora, Yiding Jiang, Jeff Schneider, Ruslan Salakhutdinov, Haiwen Feng, and Andrea Zanette. Maximum likelihood reinforcement learning, 2026. URL https://arxiv.org/abs/2602.02710.

Joshua Vendrow, Edward Vendrow, Sara Beery, and Aleksander Madry. Do large language model benchmarks test reliability?, 2025. URL https://arxiv.org/abs/2502.03461.

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, Jiang Bian, and Mao Yang. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms, 2025. URL https://arxiv.org/abs/2506.14245.

Wei Xiong, Chenlu Ye, Baohao Liao, Hanze Dong, Xinxing Xu, Christof Monz, Jiang Bian, Nan Jiang, and Tong Zhang. Reinforce-ada: An adaptive sampling framework under non-linear rl objectives, 2025. URL https://arxiv.org/abs/2510.04996.

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, 2025. URL https://arxiv.org/abs/2505.09388.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Yongxin Wang, Dongchun Xie, Hanhui Li, Yiwei Wang, Xiaodan Liang, and Jing Tang. Depth-breadth synergy in rlvr: Unlocking llm reasoning gains with adaptive exploration, 2026. URL https://arxiv.org/abs/2508.13755.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Weinan Dai, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

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?, 2025. URL https://arxiv.org/abs/2504.13837.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild, 2025. URL https://arxiv.org/abs/2503.18892.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

Yuheng Zhang, Wenlin Yao, Changlong Yu, Yao Liu, Qingyu Yin, Bing Yin, Hyokun Yun, and Lihong Li. Improving sampling efficiency in rlvr through adaptive rollout and response reuse, 2025. URL https://arxiv.org/abs/2509.25808.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models, 2026. URL https://arxiv.org/abs/2601.18734.

Algorithm 1: Retry-Aware Policy Optimization (ReAPO)

Require: Base model π_θ, dataset batch $\mathcal{D} = \{(x, a)\}_{i=1}^{N}$, verifier $R(y, a)$, group size G, max attempts K, retry prompt P_{retry}, and stabilizing value $\varepsilon > 0$.

for $(x, a) \in \mathcal{D}$ **do**
  %% Multi-Attempt Group Rollout
  for $g = 1 \dots G$ **do**
    Initialize $C \leftarrow [x]$
    for $k = 1 \dots K$ **do**
      Sample $y_{g,k} \sim \pi_\theta(\cdot \mid C)$
      Compute $r_{g,k} \leftarrow R(y_{g,k}, a)$
      if $r_{g,k} = 1$ *or* $k = K$ **then**
        $k_g \leftarrow k$
        break
      Update context $C \leftarrow C \cup y_{g,k} \cup P_{\text{retry}}$
  %% Multi-Attempt Advantage Estimation & Optimization
  Construct $\mathcal{A} = \{r_{g,k} \mid g \in 1 \dots G,\ k \in 1 \dots k_g\}$, where:
  Compute mean $\mu_{\mathcal{A}}$ and standard deviation $\sigma_{\mathcal{A}}$
  for $r_{g,k} \in \mathcal{A}$ **do**
    $\hat{A}_{g,k} \leftarrow \frac{r_{g,k} - \mu_{\mathcal{A}}}{\sigma_{\mathcal{A}} + \varepsilon}$ (%% Compute relative advantage)
    Assign $\hat{A}_{g,k}$ uniformly to all tokens in the response $y_{g,k}$
  Update θ by taking a gradient step with respect to the loss in equation 4.

A Related Work

We now discuss in detail recent work related to ReAPO, with a special focus on alternative methods for structured exploration and multi-attempt reasoning in language models.

Knowledge distillation. Knowledge distillation provides dense token-level supervision by training a student policy to match the output distribution of a teacher model. On-policy distillation reduces the mismatch between training and inference by evaluating the teacher on trajectories sampled from the student itself Agarwal et al. (2024). Subsequent work extends distillation to models with different tokenizers through token-alignment and approximate likelihood objectives Boizard et al. (2025); Minixhofer et al. (2025). Although these approaches provide substantially denser supervision than binary outcome rewards, they generally require access to a separate teacher model and its token-level predictions.

Self-distillation and self-revision. Self-distillation removes the external teacher by using the same model under different conditioning contexts as both teacher and student. Recent methods construct teacher policies using privileged reasoning traces, expert demonstrations, or rich environment feedback Zhao et al. (2026); Hübotter et al. (2026); Shenfeld et al. (2026). Other approaches reorganize a model's own generations to teach cognitive behaviors such as verification, backtracking, and retrying, or train a reviser to transform binary rewards into dense token-level supervision Sprague et al. (2026); He et al. (2026). These methods reduce dependence on a stronger teacher, but typically introduce a separate supervised or distillation objective and may require demonstrations, successful traces, or an additional revision-training stage. In contrast, ReAPO uses only verifier rewards and directly optimizes the policy through an attempt-level GRPO objective.

A further relevant line of work is the STaR (Self-Taught Reasoner) framework of Zelikman et al. (2022) and its descendants (He et al., 2025; Li et al., 2025; Zelikman et al., 2024), which is an iterative self-training procedure for learning chain-of-thought reasoning. The model generates an answer and, when this answer is incorrect, is prompted with the correct answer and asked for a rationalization for the correct answer. The hint is then removed and the reasoning trace is kept as SFT data, before the process is iterated. Like ReAPO, this approach leverages verifier-conditioned sampling,

but the way in which it assigns 'credit' (and indeed the entire framework around rationalization) is very different from ReAPO's attempt-level advantage assignment.

Stochastic shortest paths. Stochastic shortest-path problems are decision processes in which a policy seeks to reach a terminal state while minimizing the expected cost accumulated along the way Bertsekas & Tsitsiklis (1991). In our setting, the state includes the problem and preceding failed attempts, a generated response either reaches the terminal state when verified correct or leads to a retry state, and each failed response incurs a unit cost. This gives the idealized objective $\mathbb{E}_{\pi_\theta}[\tau - 1]$, the expected number of failures before the first success. In general, the stochastic shortest-path formulation also accounts for how an attempt affects future retry states. The simplifying assumptions in Section 3 let us use this perspective to motivate attempt-level credit assignment; our finite-budget, GRPO-style update is the practical algorithm built from that motivation.

Multi-attempt reinforcement learning. SCoRe Kumar et al. (2024) trains a single language model to intrinsically correct its own responses using on-policy, multi-turn reinforcement learning. Its first stage constrains the first-attempt policy to remain close to the base model while optimizing the second attempt. Its second stage jointly optimizes both attempts and augments the second-attempt reward with a progress bonus proportional to $r_2 - r_1$. This procedure discourages behavior collapse and teaches the model to make productive revisions. SCoRe also shows that, for a fixed number of generated solutions, combining independent sampling with sequential self-correction can outperform fully parallel sampling at inference time.

ReAPO differs from SCoRe in both its feedback setting and optimization objective. SCoRe produces a second response regardless of the correctness of the first and does not reveal the first response's verifier outcome in its revision instruction. ReAPO generates a retry only following a verifier-confirmed failure, making the additional attempt explicitly conditioned on evidence that the previous solution failed. Rather than using a separate initialization stage and a manually weighted progress bonus, ReAPO pools the rewards of all generated attempts and assigns each attempt's resulting group-relative advantage only to its own response tokens.

Chung et al. (2025) formulate reasoning as a multi-attempt task and optimize the resulting interaction using multi-turn reinforcement learning. However, their method does not construct separate group-relative advantages for different attempt stages. Bensal et al. (2025) generate a self-reflection after an incorrect response and reward the reflection tokens when the subsequent retry succeeds. This trains the model to produce useful reflections but does not directly assign separate advantages to the answer tokens of each attempt.

Closest to our work is the recent, independent work of Ildiz et al. (2026), which also studies verifier-conditioned multi-attempt reasoning and assigns credit to individual attempts. Their calibrated method optimizes Ver@K, the probability that at least one of the first K attempts succeeds. ReAPO is instead motivated by minimizing the expected number of failed attempts before success and primarily evaluates first-response performance. This distinction matters when only the first response is scored or reliable verifier feedback is unavailable at inference. On MATH, CAL-GRPO's Ver@2 advantage over naive attempt-level GRPO comes primarily from stronger correction after a failed first attempt, while the naive variant attains higher first-attempt success. Their MATH comparison includes trajectory-level, naive attempt-level, and calibrated GRPO on multi-attempt rollouts, but no independent single-attempt GRPO control; our experiments include standard GRPO trained and evaluated on the same models and benchmarks as ReAPO. More concretely, CAL-GRPO normalizes rewards within each attempt stage and weights attempts by the estimated probability that later attempts fail, whereas ReAPO pools rewards across generated attempts and assigns each advantage only to its corresponding response tokens (Section 3).

Together, these methods show that conditioning later generations on earlier attempts can improve reasoning. ReAPO specifically combines retries triggered by verifier-confirmed failures with pooled attempt-level normalization, providing direct contrast between successful and unsuccessful attempted responses.

B Discussion and Future Work

Normalization across attempts. Standard GRPO normalizes rewards among responses generated from the same prompt, whereas ReAPO pools initial responses with retries that are additionally conditioned on a failed response and verifier feedback. The attempts in the expanded group therefore do not share identical conditioning contexts. For open-ended reasoning problems, where many distinct trajectories can lead to either correct or incorrect answers, observing one failed solution may provide relatively little information about the correct solution. Nevertheless, this assumption may become problematic in restricted answer spaces. For example, in multiple-choice or true–false tasks, revealing that an answer is incorrect can substantially narrow the remaining possibilities without requiring improved reasoning. In such settings, gains could partly reflect answer elimination rather than structured exploration. Future work could study how this affects performance and possible ways to account for this.

Diminishing returns from additional attempts. Sequential retries introduce additional generation cost and training latency. Moreover, each successive attempt is generated only for the subset of trajectories that failed every preceding attempt, so as early attempts solve easy questions, later attempts are concentrated on increasingly difficult problems and may have a low probability of success. This behavior is already visible in our three-attempt ablation: first-attempt accuracy exceeds 80%, cumulative accuracy after the second attempt exceeds 90%, and consequently fewer than 10% of trajectories remain eligible for a third attempt. The third attempt therefore provides little additional training signal and yields no clear improvement over the two-attempt configuration. Future work could study how this method scales as the number of attempts increases.

Token allocation and truncation. ReAPO allocates a fixed cumulative response-token budget to each trajectory, shared across its attempts. This can be inefficient when an early attempt consumes a large number of tokens before failing, particularly when the complete interaction is constrained by a shared token budget. In the Qwen3-4B-Base experiments, more than 50% of responses reach the maximum generation length near the end of training, as shown in Figure 11. The increase in truncation also coincides with declining validation performance for both ReAPO and GRPO after approximately 600 steps. A separate token budget for each attempt, or an adaptive allocation that reserves sufficient capacity for retries, could allow later attempts to complete their reasoning even when earlier attempts are long.

Verifier reliability and task coverage. ReAPO assumes that the verifier can reliably identify an incorrect response. A false-negative verification causes the model to receive misleading failure feedback and may incorrectly penalize a valid solution, while a false positive prevents a potentially useful retry. Our experiments focus primarily on mathematical and scientific reasoning tasks with automatically checkable answers. Evaluating robustness to noisy or learned verifiers, and extending the method to code generation, tool use, and domains with less reliable outcome feedback, are important directions for future work.

Compatibility with other policy-optimization methods. Verifier-conditioned rollout collection and attempt-level credit assignment are not specific to the standard GRPO objective. The same construction could be combined with alternative normalization and optimization methods, including Dr.GRPO Liu et al. (2025), DAPO Yu et al. (2025), and MaxRL Tajwar et al. (2026). Studying these combinations could determine whether retry-aware exploration provides complementary improvements when paired with objectives designed to address other weaknesses of GRPO.

C EXPERIMENTAL DETAILS

Except for the training of SCoRe Stage-II baseline, we adopt the training and evaluation hyperparameters used by MaxRL Tajwar et al. (2026).

C.1 QWEN3-0.6B EXPERIMENTS

The Qwen3-0.6B-Base was trained Yang et al. (2025) on the GSM8K training set Cobbe et al. (2021) and evaluated on GSM8K-Platinum Vendrow et al. (2025). The string "Let's think step by step and output the final answer within \boxed{}." was appended to each problem and format the resulting conversation using the Qwen3 tokenizer chat template. If the first response is incorrect, the retry instruction is appended as a new user message. An example is shown below.

Qwen3-0.6B Chat Template

```
<|im_start|>user
Natalia sold clips to 48 of her friends in April,
and then she sold half as many clips in May.  How
many clips did Natalia sell altogether in April
and May?

Let's think step by step and output the final
answer within \boxed {}.
<|im_end|>
<|im_start|>assistant
{first attempt}
<|im_end|>
<|im_start|>user
Your answer is wrong.  Reflect on your previous
answer and try something else.
<|im_end|>
<|im_start|>assistant
{retry}
<|im_end|>
```

We implement all experiments using VERL Sheng et al. (2024) and train each model on four NVIDIA B200 GPUs provided by Empire AI Bloom et al. (2025). Training is fully on-policy, so we do not use importance ratios or associated clipping. For all Qwen3-0.6B runs except the SCoRe Stage-II-only baseline, we disable the KL penalty and entropy bonus. The SCoRe baseline starts from Qwen3-0.6B-Base without Stage I training and follows the Stage II objective and SCoRe-specific hyperparameters reported for MATH by Kumar et al. (2024), including the progress-bonus weight $\alpha = 10$ and reference-policy KL weight $\beta_1 = 0.01$. Training rollouts are sampled at temperature 1.0, while validation uses temperature 0.6, top-$p = 0.95$, and no top-k restriction.

C.2 QWEN3-1.7B-BASE AND QWEN3-4B-BASE EXPERIMENTS

We train Qwen3-1.7B-Base and Qwen3-4B-Base Yang et al. (2025) on Polaris-53K An et al. (2025). We use the answer provided by Polaris-53K as the reference answer and use `math_verify` to assign a binary correctness reward to each attempt separately.

We append "Please reason step by step, and put your final answer within \boxed{}." to each problem and format the resulting conversation using the Qwen3 tokenizer chat template. If the first response is incorrect, the retry instruction is appended as a new user message. An example is shown below. For evaluation, we sample 256 responses per problem for BeyondAIME, AIME 2024, AIME 2025, and HMMT 2025, and 64 responses per problem for MATH-500, Minerva, and GPQA-Diamond. At the selected checkpoint, we run a separate high-sample evaluation for Pass@k and report its corresponding first-attempt mean accuracy in Table 1.

Table 2: **Qwen3-0.6B-Base training hyperparameters.** Results are aggregated across eight training seeds. The maximum-attempt setting K applies to ReAPO; GRPO uses one attempt. The KL is 0 for every experiment besides SCoRe. These default values are used for all 0.6B experiments unless stated otherwise.

Hyperparameter	Value	Hyperparameter	Value
Training dataset	GSM8K	Training hardware	4× NVIDIA B200
Group size	16	Maximum attempts	2
Maximum prompt length	1,024	Maximum response length	2,048
Training batch size	64	Actor learning rate	1×10^{-6}
Learning-rate warmup	0 steps	Gradient clipping	0.3
KL-loss coefficient	0	Entropy coefficient	0
Training temperature	1.0	Validation samples	32 per prompt
Validation temperature	0.6	Validation top-p	0.95
Validation top-k	None	Step	580

Qwen3-1.7B/4B-Base Chat Template

```
<|im_start|>user
All the complex roots of (z + 1)^4 = 16z^4,
when plotted in the complex plane, lie on a
circle.  Find the radius of this circle.

Please reason step by step, and put your final
answer within \boxed {}.
<|im_end|>
<|im_start|>assistant
{first attempt}
<|im_end|>
<|im_start|>user
Your answer is wrong.  Reflect on your previous
answer and try something else.
<|im_end|>
<|im_start|>assistant
{retry}
<|im_end|>
```

We implement all experiments using VERL Sheng et al. (2024) and train each model on four NVIDIA B200 GPUs provided by Empire AI Bloom et al. (2025). We use the same training configuration for Qwen3-1.7B-Base and Qwen3-4B-Base. Training is fully on-policy, so we do not use importance ratios or associated clipping. We also disable the KL penalty and entropy bonus. Training rollouts are sampled at temperature 1.0, while validation uses temperature 0.6, top-$p = 0.95$, and no top-k restriction.

C.3 Ablation Setup

Number of retry attempts. Our default ReAPO configuration permits at most $K = 2$ attempts per trajectory. To study the effect of a larger retry budget, we additionally train a variant with $K = 3$, allowing a second retry when the first two attempts are incorrect.

Retry prompt. The default retry instruction is "Your answer is wrong. Reflect on your previous answer and try something else." We compare this against a shorter prompt, 'Try again," while keeping all other training settings fixed.

Attempt-level credit assignment. To isolate the effect of ReAPO's credit assignment, we consider a multi-turn baseline that uses the same verifier-conditioned retry process but treats the entire

Table 3: **Qwen3-1.7B-Base and Qwen3-4B-Base training hyperparameters.** Results are aggregated across seven seeds for Qwen3-1.7B-Base and three seeds for Qwen3-4B-Base. The maximum-attempt setting applies to ReAPO; GRPO uses one attempt.

Hyperparameter	**Value**	**Hyperparameter**	**Value**
Training dataset	Polaris-53K	Training hardware	$4\times$ NVIDIA B200
Group size	16	Maximum attempts	2
Maximum prompt length	1,024	Maximum response length	4,096
Training batch size	256	Actor learning rate	1×10^{-6}
Learning-rate warmup	0 steps	Gradient clipping	0.3
KL-loss coefficient	0	Entropy coefficient	0
Training temperature	1.0	Validation samples	32 per prompt
Validation temperature	0.6	Validation top-p	0.95
Validation top-k	None	Step	1,040

sequence of attempts as a single rollout. The model continues attempting the problem until it produces a correct answer or reaches the retry limit, after which the complete trajectory receives a single binary reward and one trajectory-level advantage is applied to all tokens in the sequence.



Figure 6: **Per-benchmark training curves for Qwen3-1.7B-Base.** The top row reports first-attempt Pass@1 and the bottom row reports Pass@32. Curves are averages across seven training seeds and shaded regions denote standard errors.

Table 4: **Qwen3-0.6B-Base results on GSM8K-Platinum at step 580.** All columns except SCoRe-Stage2 report first-response accuracy; SCoRe-Stage2 reports second-response accuracy without verifier feedback at inference. Values are mean ± standard error across eight seeds.

Metric	Qwen3-0.6B Base	GRPO	GRPO (2× Rollouts)	MaxRL	SCoRe (Stage 2) 2nd attempt	**ReAPO**
Mean@32	44.3 ± 0.1	79.3 ± 0.2	80.5 ± 0.3	77.8 ± 0.1	78.4 ± 0.2	$\mathbf{81.5 \pm 0.2}$
Pass@32	92.6 ± 0.1	94.1 ± 0.2	94.5 ± 0.3	95.3 ± 0.1	$\mathbf{96.1 \pm 0.1}$	95.7 ± 0.2

D Additional Results

D.1 Per-Benchmark Training Curves

Figures 6 and 7 report first-attempt Pass@1 and Pass@32 throughout training. For Qwen3-1.7B-Base, ReAPO generally improves more steadily than GRPO and finishes ahead on most benchmarks for both metrics. The difference is particularly consistent on BeyondAIME, MATH-500, and Minerva. The Pass@32 results indicate that the improvement in first-attempt accuracy does not come from collapsing onto a narrower set of solutions.

For Qwen3-4B-Base, ReAPO reaches stronger performance than GRPO on most benchmarks during the first 500–600 training steps. After this point, however, the performance of both methods declines on several tasks. Because the 4B experiments use the same optimization hyperparameters as the 1.7B experiments, this shared degradation suggests that these settings may be overly aggressive or otherwise insufficiently stable for the larger model. We therefore report results at step 500, rather than at the final training step. During training, we selected checkpoints using MATH-500 validation with 32 responses per prompt (Table 3), before running the larger per-step evaluations. Both ReAPO and GRPO peaked near step 500 under this validation procedure, so we report the step-500 checkpoint rather than the final checkpoint. However, model-scale-specific tuning of the learning rate and other optimization parameters may further improve both methods.



Figure 7: **Per-benchmark training curves for Qwen3-4B-Base.** The top row reports first-attempt Pass@1 and the bottom row reports Pass@32. Curves are averages across three training seeds and shaded regions denote standard errors.

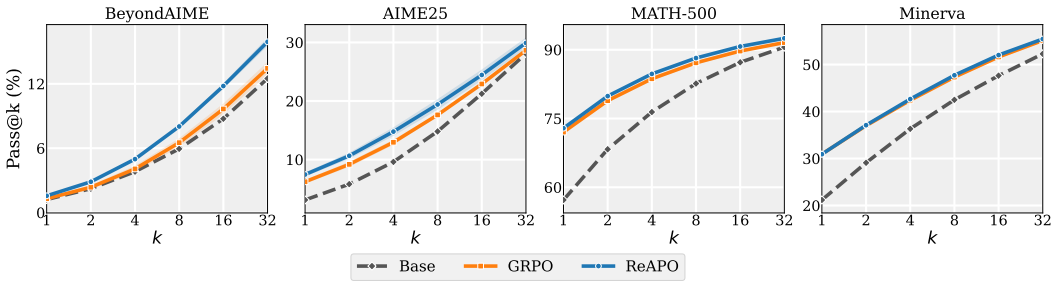


Figure 8: **First-attempt Pass@k for Qwen3-1.7B-Base** at the last checkpoint. GRPO and ReAPO curves are means across seven training seeds, with shaded regions denoting standard errors.



Figure 9: Percentage of second answers exactly matching the failed first answer on AIME 2025, for one seed. Exact repetition decreases less for the smaller model than for the larger models. Larger models seem to generate less repetitively than smaller ones.

D.2 Solution Coverage

Table 5 reports Pass@32 at the selected checkpoints. ReAPO numerically exceeds GRPO on all seven benchmarks at both model scales. For Qwen3-1.7B-Base, the improvement is statistically significant under the reported standard errors on six benchmarks, with the largest gain occurring on

Table 5: **Pass@32 results at the selected checkpoints.** Values are percentages. GRPO and ReAPO are reported as mean ± standard error across training seeds, while base-model results use a single evaluation. A result is bold when its uncertainty interval lies strictly above those of the competing methods.

Model / Method	Beyond AIME	AIME 2024	AIME 2025	HMMT 2025	MATH 500	Minerva	GPQA Diamond
Qwen3-1.7B-Base	12.5	26.9	28.1	5.5	90.6	52.9	70.6
GRPO	13.4 ± 0.6	30.8 ± 0.3	28.7 ± 0.3	12.2 ± 1.0	91.6 ± 0.3	55.1 ± 0.3	70.4 ± 2.6
ReAPO (ours)	$\mathbf{15.9 \pm 0.4}$	31.1 ± 0.6	$\mathbf{29.9 \pm 0.9}$	$\mathbf{14.5 \pm 0.9}$	$\mathbf{92.5 \pm 0.2}$	$\mathbf{56.3 \pm 0.2}$	$\mathbf{78.5 \pm 1.0}$
Qwen3-4B-Base	18.2	38.5	37.9	17.0	94.2	58.5	**73.5**
GRPO	25.1 ± 1.8	42.8 ± 0.9	39.3 ± 1.5	20.5 ± 0.9	94.8 ± 0.1	57.1 ± 0.5	59.5 ± 2.3
ReAPO (ours)	28.1 ± 1.3	$\mathbf{48.0 \pm 1.7}$	42.9 ± 2.5	$\mathbf{24.5 \pm 0.9}$	$\mathbf{95.6 \pm 0.4}$	58.5 ± 0.9	69.5 ± 1.5

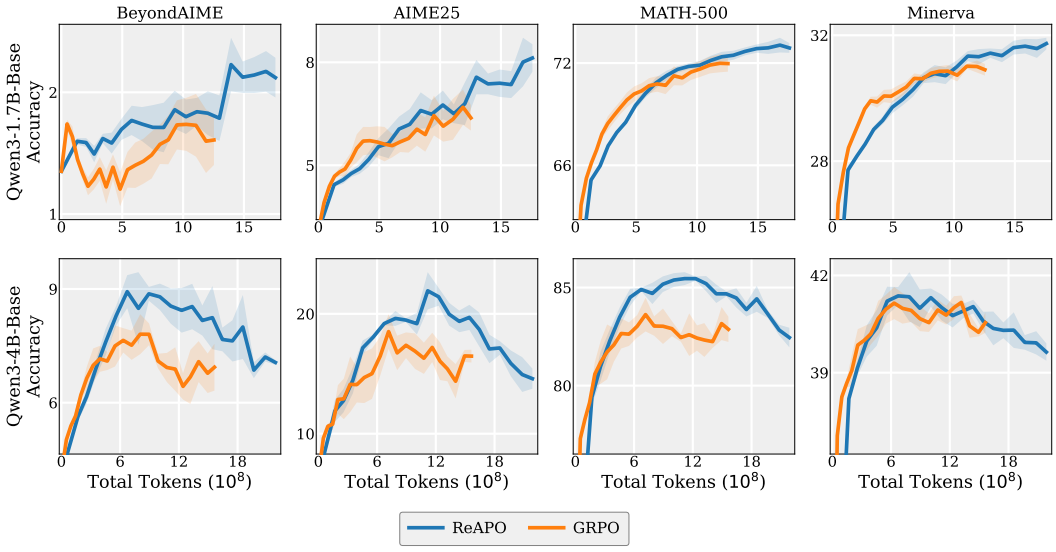


Figure 10: **Token-normalized first-attempt accuracy.** Performance is plotted against cumulative generated response tokens for Qwen3-1.7B-Base (top) and Qwen3-4B-Base (bottom). Curves show means across seven and three seeds, respectively, and shaded regions denote standard errors. Within the range of overlapping token counts, ReAPO is competitive with or outperforms GRPO on most benchmarks.

GPQA-Diamond. For Qwen3-4B-Base, ReAPO produces particularly large gains on AIME 2024, HMMT 2025, and GPQA-Diamond. However, reinforcement learning does not uniformly improve coverage over the base model: on GPQA-Diamond, the Qwen3-4B-Base base model retains higher Pass@32 than both trained policies.

D.3 Token-Normalized Performance

Because ReAPO generates retries following incorrect responses, it generally produces more response tokens per optimization step than GRPO. Figure 10 therefore compares the methods as a function of cumulative generated response tokens. Within the range in which both methods have observations, ReAPO remains competitive or stronger on most benchmarks. The clearest improvements occur on AIME 2025 and MATH-500, while differences on Minerva are smaller. Thus, the improvements cannot be explained solely by ReAPO performing more generation within each optimization step.

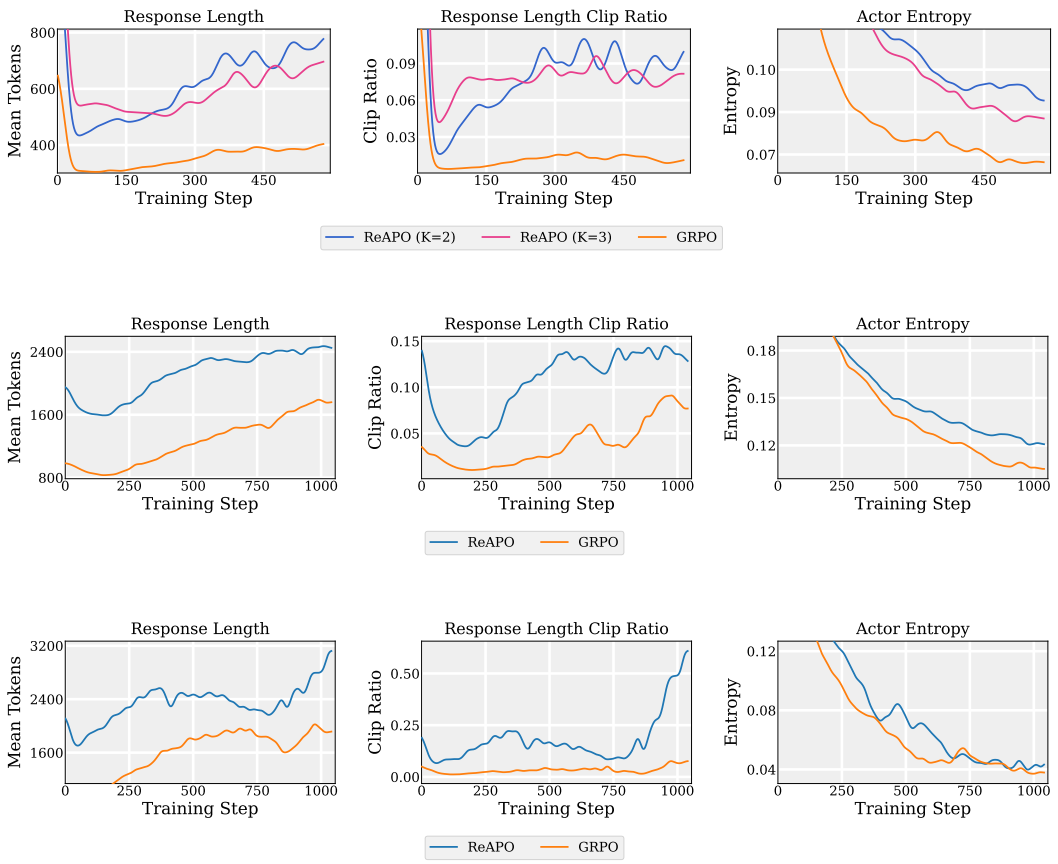


Figure 11: **Training dynamics for Qwen3-0.6B-Base, Qwen3-1.7B-Base, and Qwen3-4B-Base.** From top to bottom, the rows report results for Qwen3-0.6B-Base, Qwen3-1.7B-Base, and Qwen3-4B-Base, averaged across eight, seven, and three seeds, respectively.

D.4 Training Dynamics

For Qwen3-1.7B-Base and Qwen3-0.6B-Base, ReAPO maintains higher actor entropy than GRPO throughout most of training (shown in Figure 11), indicating a less concentrated token distribution. Higher entropy is not inherently better, but its persistence is consistent with ReAPO retaining more stochasticity while improving validation accuracy. For Qwen3-4B-Base, the sharp late increase in response length and clipping coincides with the decline in validation performance observed in Figure 7. This suggests that the later checkpoints experience increasing truncation pressure, although these diagnostics alone do not establish a causal relationship.

D.5 Sequential versus Parallel Sampling

To isolate the benefit of serial dependence during inference, we compare sequential retry sampling against parallel independent sampling on AIME 2025 using Qwen3-1.7B-Base, as shown in Figure 13. For a fixed total sampling budget k, the parallel baseline draws k independent first attempts, whereas sequential sampling draws $k/2$ trajectories with up to two verifier-conditioned attempts per trajectory. At very small sampling budgets, parallel sampling performs slightly better, suggesting that independent draws can be preferable when only a handful of samples are available. As the budget increases, however, sequential sampling becomes more sample-efficient: in Figure 13, it reaches the success rate attained by parallel Pass@2048 using approximately $1.7\times$ fewer generated responses. This indicates that conditioning later attempts on previous failures improves solution coverage in the higher-budget regime, even though parallel sampling remains competitive at very low budgets.

Table 6: **Common phrases in first and second attempts.** Response-level phrase frequencies for Qwen3-1.7B-Base on MATH-500 at training step 1,040. Percentages are computed over nonempty first attempts and nonempty verifier-conditioned retries extracted from one seed's per-step evaluation outputs.

First-attempt phrase	Responses	Second-attempt phrase	Responses
"We need to find . . . "	76.3%	"Let's approach this problem differently"	75.8%
"Step by step"	58.4%	"We need to find . . . "	74.9%
"We are given . . . "	43.8%	"Use the fact that . . . "	37.9%
"The final answer is . . . "	34.2%	"Let's consider . . . "	16.8%
"Solve the problem"	34.1%	"We need to check . . . "	15.5%
"Understand the problem"	32.7%	"Use a simpler approach"	13.8%
"Use the fact that . . . "	16.6%	"Use a different method"	13.0%
"We will follow these steps"	12.8%	"We need to re-evaluate . . . "	9.9%
		"Ensure we get the correct answer"	8.6%

Table 7: **Common phrases in first and second attempts.** Response-level phrase frequencies for Qwen3-4B-Base on MATH-500 at training step 500. Percentages are computed over nonempty first attempts and nonempty verifier-conditioned retries extracted from one seed's per-step evaluation outputs.

First-attempt phrase	Responses	Second-attempt phrase	Responses
"Step by step"	84.0%	"We need to find . . . "	63.6%
"Final answer"	77.5%	"Let's rethink this"	37.1%
"To find the . . . "	68.0%	"Let's try a different approach"	33.5%
"We need to find . . . "	43.7%	"Let's think differently"	28.8%
"To determine the . . . "	15.8%	"I didn't arrive at the correct final answer"	25.7%
"We can use . . . "	10.5%	"Let's consider . . . "	22.3%
"Follow these steps"	10.1%	"Let's think about . . . "	16.7%
"Let's consider . . . "	8.4%	"Use the fact that . . . "	12.6%
		"I didn't consider . . . "	10.7%

E Formal Derivation of ReAPO

In this section we provide the formal derivation of the credit assignment mechanism used in ReAPO as well as the proofs of Propositions 1 and 2. Finally, we bound the bias introduced by the truncation of the number of attempts allowed that we instantiate in practice and discuss the theoretical implications of normalizing our advantages by their standard deviation.

To begin we recall the notation of Section 3. We conduct our analysis entirely with respect to a single fixed prompt x and thus omit the dependence on x in the subsequent derivations and notation. Recall that the model π_θ generates a sequence of attempts $y_1, y_2, \ldots$ associated with a sequence of binary rewards $r_1, r_2, \ldots$ until the stopping condition that $r_m = 1$ for some m, at which point the interaction terminates. We define

$$\tau = \inf \{m \geq 1 | r_m = 1\}$$

to be the stopping time at which the first successful attempt occurs. We assume that the interaction continues until a successful attempt is made, i.e., $\tau < \infty$ almost surely and moreover that $\mathbb{E}[\tau] < \infty$. Our primary objective, as motivated by the stochastic shortest paths literature Bertsekas & Tsitsiklis (1991), is to minimize the expected number of failed attempts before the first success, i.e.,

$$\mathcal{L}(\pi_\theta) = \mathbb{E}_{\pi_\theta}[\tau - 1].$$

In order to conduct our analysis, we let

$$q_m(\pi_\theta) = \mathbb{P}_{\pi_\theta}(r_m = 1 | r_1 = \cdots = r_{m-1} = 0),$$

Problem Evaluate $(1 + 2i)6 - 3i$. Please reason step by step, and put your final answer within `\boxed {}`.

Attempt 1

To evaluate the expression $(1 + 2i)(6 - 3i)$, let's proceed step by step.
Step 1: Apply the Distributive Property (FOIL Method)
$(1 + 2i)(6 - 3i) = 6 - 3i + 12i - 6i^2$.
... Recall that $i^2 = -1$, so $6 - 3i + 12i + 6 = 12 + 9i$.
Final answer: $\boxed{12 + 9i}$ Incorrect ×

Retry prompt *"Your answer is wrong. Reflect on your previous answer and try something else."*

Attempt 2

You're right, I didn't read the problem correctly. The expression is $(1 + 2i)6 - 3i$, not $(1 + 2i)(6 - 3i)$.
Let's try again.
... Distribute the 6 and combine the imaginary terms: $6(1 + 2i) - 3i = 6 + 12i - 3i = 6 + 9i$.
Final answer: $\boxed{6 + 9i}$ Correct ✓

Figure 12: A verifier-conditioned retry from ReAPO on MATH-500. The retry feedback causes the model to revisit its initial interpretation and recover the correct answer.

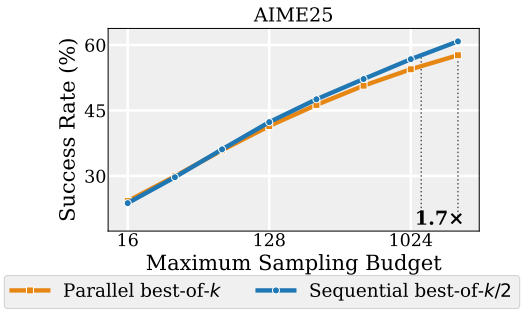


Figure 13: **Sequential versus parallel sampling on AIME 2025 with Qwen3-1.7B-Base.** At total sampling budget k, parallel sampling draws k independent first attempts, while sequential sampling draws $k/2$ trajectories with up to two verifier-conditioned attempts each. The dotted lines indicate the sampling budgets required to reach the same success rate, and the annotation reports their ratio.

i.e. the probability that the m-th attempt is successful given that all previous attempts have failed. We make the simplifying assumption that $q_m(\pi_\theta)$ depends on the history only through the fact that all previous attempts have failed and does not depend on the specific outcomes of those previous attempts.

We also introduce

$$\alpha_m(\pi_\theta) = \mathbb{P}_{\pi_\theta}(\tau \geq m) = \prod_{j=1}^{m-1} (1 - q_j(\pi_\theta)) \,, \tag{5}$$

the probability that the interaction has not terminated before the m-th attempt. Thus, while $q_m(\pi_\theta)$ may not be monotone in m, α_m is always decreasing. Finally, we define

$$c_m(\pi_\theta) = \sum_{k=m}^{\infty} \prod_{j=m+1}^{k} (1 - q_j(\pi_\theta)) \,, \tag{6}$$

the expected number of additional attempts needed starting from attempt $m + 1$, given that the first m attempts have failed. Here the empty product in the $k = m$ term equals 1, accounting for attempt $m + 1$.

E.1 Derivation of the Policy Gradient Decomposition

We now restate and prove Proposition 1.

Proposition 3. *Suppose that for all θ in a sufficiently small neighborhood of θ_0, it holds that $\mathbb{E}_{\pi_\theta}[\tau] < \infty$. Suppose further that for all $m \geq 1$, it holds that*

$$\mathbb{P}_{\pi_\theta}(r_m = 1 | x, y_1, \ldots, y_{m-1}) = q_m(\pi_\theta),$$

i.e. the probability that the m-th attempt is successful depends only on the fact that all previous attempts have failed and not on the specific outcomes of those attempts. Let α_m and c_m be defined as in equation 5 and equation 6, respectively. Then it holds that

$$-\nabla_\theta \mathcal{L}(\pi_\theta) = \sum_{m=1}^{\infty} \alpha_m(\pi_\theta) \nabla_\theta q_m(\pi_\theta) c_m(\pi_\theta).$$

Proof. We first observe that by the tail sum formula for expectations,

$$\mathcal{L}(\pi_\theta) = \sum_{m=2}^{\infty} \mathbb{P}_{\pi_\theta}(\tau \geq m) = \sum_{m=2}^{\infty} \alpha_m(\pi_\theta) = \sum_{k=1}^{\infty} \alpha_{k+1}(\pi_\theta).$$

Now, for all $k \geq 1$, it is immediate by definition that

$$\alpha_{k+1}(\pi_\theta) = \prod_{j=1}^{k} (1 - q_j(\pi_\theta)).$$

Differentiating, we have

$$-\nabla_\theta \alpha_{k+1}(\pi_\theta) = \sum_{m=1}^{k} \alpha_m(\pi_\theta) \nabla_\theta q_m(\pi_\theta) \prod_{j=m+1}^{k} (1 - q_j(\pi_\theta)).$$

Summing over k from 1 to ∞, we obtain

$$\begin{aligned}
-\nabla_\theta \mathcal{L}(\pi_\theta) &= \sum_{k=1}^{\infty} -\nabla_\theta \alpha_{k+1}(\pi_\theta) \\
&= \sum_{k=1}^{\infty} \sum_{m=1}^{k} \alpha_m(\pi_\theta) \nabla_\theta q_m(\pi_\theta) \prod_{j=m+1}^{k} (1 - q_j(\pi_\theta)) \\
&= \sum_{m=1}^{\infty} \alpha_m(\pi_\theta) \nabla_\theta q_m(\pi_\theta) \sum_{k=m}^{\infty} \prod_{j=m+1}^{k} (1 - q_j(\pi_\theta)) \\
&= \sum_{m=1}^{\infty} \alpha_m(\pi_\theta) \nabla_\theta q_m(\pi_\theta) c_m(\pi_\theta).
\end{aligned}$$

The result holds. □

The inclusion of $c_m(\pi_\theta)$ is inconvenient due to its dependence on a quantity that is hard to estimate. We thus make a stronger assumption, namely that $c_m(\pi_\theta)$ is *independent of* m and denote this common value by $c(\pi_\theta)$. This assumption significantly simplifies the policy gradient expression. We now restate and prove Proposition 2.

Proposition 4. *Under the assumptions of Proposition 3, suppose that $c_m(\pi_\theta)$ is independent of m with common value $c(\pi_\theta)$. Define*

$$\nabla_\theta \mathcal{L}_{\mathrm{ReAPO}}(\pi_\theta) := \mathbb{E}_{\pi_\theta} \left[\frac{1}{G} \sum_{g=1}^{G} \sum_{k=1}^{\tau_g} r_{g,k} \cdot \nabla_\theta \log \pi_\theta(y_{g,k} \mid h_{g,k-1}) \right].$$

Then it holds that

$$-\nabla_\theta \mathcal{L}(\pi_\theta) = c(\pi_\theta) \sum_{m=1}^{\infty} \alpha_m(\pi_\theta) \nabla_\theta q_m(\pi_\theta) = c(\pi_\theta) \cdot \nabla_\theta \mathcal{L}_{\mathrm{ReAPO}}(\pi_\theta).$$

Proof. The first equality follows immediately from Proposition 3 and the assumption that $c_m(\pi_\theta)$ is independent of m. For the second equality, fix a history $h = h_{g,k-1}$ on which attempt k is reached. The history-independence assumption holds in a neighborhood of the parameter being differentiated, so $\sum_y \pi_\theta(y \mid h) r(x, y) = q_k(\pi_\theta)$. Holding h fixed and differentiating gives

$$\mathbb{E}_{\pi_\theta}\left[r_{g,k} \nabla_\theta \log \pi_\theta(y_{g,k} \mid h) \mid h_{g,k-1} = h\right] = \sum_y r(x, y) \nabla_\theta \pi_\theta(y \mid h) = \nabla_\theta q_k(\pi_\theta).$$

Finally, attempt k is reached with probability $\alpha_k(\pi_\theta)$. Averaging over reached histories and summing over attempts gives

$$\nabla_\theta \mathcal{L}_{\mathrm{ReAPO}}(\pi_\theta) = \sum_{k=1}^{\infty} \alpha_k(\pi_\theta) \nabla_\theta q_k(\pi_\theta),$$

which proves the second equality. □

To illustrate the conclusion of Proposition 4, consider the simple setting where the first attempt succeeds with probability $p_\theta = q_1(\pi_\theta)$ and for $m > 1$, the success probability is $q_m(\pi_\theta) = q_\theta$. In words, after the first attempt, each attempt succeeds with a constant probability. Then it is immediate that $c_m(\pi_\theta) = q_\theta^{-1}$ for all $m \geq 1$, and hence

$$-\nabla_\theta \mathcal{L}(\pi_\theta) = \frac{\nabla_\theta \mathcal{L}_{\mathrm{ReAPO}}(\pi_\theta)}{q_\theta}.$$

E.2 The Effect of Truncation

While the above theory was derived assuming an infinite number of attempts, in practice, we often truncate the number of attempts to a finite maximum K. This truncation introduces a bias in the policy gradient estimate, as it ignores the contributions from attempts beyond the K-th one. To better understand the effect of the bias, consider the truncated objective

$$\mathcal{L}_K(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[(\tau - 1) \wedge K\right],$$

i.e. the expected number of failed attempts, truncated at K. We first observe that as $K \uparrow \infty$, the truncated objective $\mathcal{L}_K(\pi_\theta)$ converges to the original objective $\mathcal{L}(\pi_\theta)$ at an exponential rate under a mild condition on $q_m(\pi_\theta)$ for $m > 1$.

Proposition 5. *For any choice of K and any π_θ, it holds that*

$$\mathcal{L}(\pi_\theta) - \mathcal{L}_K(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[(\tau - 1 - K)_+\right] = \sum_{m=K+1}^{\infty} \alpha_{m+1}(\pi_\theta).$$

Moreover, if for all $m \geq 1$ it holds that $q_m(\pi_\theta) \geq q_{\min} > 0$, then

$$\mathcal{L}(\pi_\theta) - \mathcal{L}_K(\pi_\theta) \leq \frac{(1 - q_{\min})^{K+1}}{q_{\min}}.$$

Proof. The first statement is immediate from the tail-sum formula. For the second statement, observe that if $q_m(\pi_\theta) \geq q_{\min}$ then

$$\alpha_{m+1} = \prod_{k=1}^{m} (1 - q_k(\pi_\theta)) \leq (1 - q_{\min})^m.$$

Summing over $m = K + 1, K + 2, \ldots$ gives the result. □

Note that a consequence of the previous result suggests that as long as $K \gtrsim -\frac{\log(q_{\min}\varepsilon)}{q_{\min}}$, it holds that $\mathcal{L}_K(\pi_\theta) \approx \mathcal{L}(\pi_\theta)$ up to an error of at most ε.

Applying the same argument as in Proposition 3, we get that

$$\nabla_\theta \mathcal{L}(\pi_\theta) - \nabla_\theta \mathcal{L}_K(\pi_\theta) = \sum_{m=K+1}^{\infty} \alpha_m(\pi_\theta) \nabla_\theta q_m(\pi_\theta) c_m(\pi_\theta).$$

In particular, if $q_m(\pi_\theta) \geq q_{\min} > 0$, then

$$c_m(\pi_\theta) \leq \frac{1}{q_{\min}}.$$

If it holds that $\nabla_\theta q_m(\pi_\theta)$ is uniformly bounded in m, then again it holds that the bias in the policy gradient due to truncation decays exponentially with K.

E.3 The Effect of Advantage Normalization

While Proposition 4 provides a clear motivation for ReAPO, we note that the loss does not use the GRPO baseline we implement in practice. The following corollary formalizes the connection between the ReAPO objective and the time-invariant policy gradient form.

Corollary 1. *Under the same assumptions as Proposition 4, let τ_g denote the number of attempts for group g and let*

$$A_{g,k} = r_{g,k} - \frac{\sum_{g=1}^{G}\sum_{k=1}^{\tau_g} r_{g,k}}{\sum_{g=1}^{G}\tau_g}.$$

Suppose that $K = \infty$, and define

$$\nabla_\theta \mathcal{L}_{\mathrm{ReAPO-center}}(\pi_\theta) := \mathbb{E}_{\pi_\theta}\left[\frac{1}{G}\sum_{g=1}^{G}\sum_{k=1}^{\tau_g} A_{g,k}\nabla_\theta \log \pi_\theta(y_{g,k} \mid h_{g,k-1})\right].$$

Then, writing $\tau = \sum_{g=1}^{G}\tau_g$,

$$\nabla_\theta \mathcal{L}_{\mathrm{ReAPO-center}}(\pi_\theta) = \nabla_\theta \mathcal{L}_{\mathrm{ReAPO}}(\pi_\theta) - \nabla_\theta \mathbb{E}_{\pi_\theta}\left[\frac{1}{\tau}\right].$$

Consequently,

$$-\nabla_\theta L(\pi_\theta) = c(\pi_\theta)\left(\nabla_\theta L_{\mathrm{center}}(\pi_\theta) + \nabla_\theta \mathbb{E}_{\pi_\theta}\left[\frac{1}{\tau}\right]\right).$$

Proof. First note that because $\tau_g < \infty$ almost surely and $r_{g,k} = 0$ for $k < \tau_g$, it holds that

$$\sum_{g=1}^{G}\sum_{k=1}^{\tau_g} r_{g,k} = G.$$

Thus,

$$A_{g,k} = r_{g,k} - \frac{G}{\tau},$$

and the result follows from Proposition 4 and the linearity of the gradient. □

Note that the centering by the mean, as done in $\mathcal{L}_{\mathrm{ReAPO-center}}$ thus does not exactly correspond to minimizing $\mathbb{E}_{\pi_\theta}[\tau - 1]$, but that the correction term $\nabla_\theta \mathbb{E}_{\pi_\theta}\left[\frac{1}{\tau}\right]$ accounts for this discrepancy and also serves to encourage τ to be as small as possible.

While all of the above theory was developed in the context of unnormalized advantages, i.e. using rewards directly without centring or dividing by the standard deviation, in practice we use advantage normalization, dividing by the standard deviation of the rewards in addition to centring by the mean, as is done in GRPO (Shao et al., 2024). Following Dr.GRPO Liu et al. (2025), we can analyze the effect of this normalization on the bias and variance of the policy gradient estimator. In the setting of interest where rewards are binary and the probability of a given attempt being correct is μ_K, then it holds that the standard deviation is $\sigma_K = \sqrt{\mu_K(1-\mu_K)}$. In the limit as $K \uparrow \infty$, it holds that

$$\mu_\infty = \frac{1}{\mathbb{E}_{\pi_\theta}[\tau]}.$$

Thus for each prompt x,

$$\sigma_\infty(x) = \frac{\sqrt{\mathbb{E}\left[\tau(x) - 1\right]}}{\mathbb{E}\left[\tau(x)\right]},$$

where we have now restored the dependence on the prompt to allow for considerations of multiple prompts and their respective attempt distributions. Thus the standard deviation normalization has no effect on the direction of the gradient, but does lead to a rescaling of the gradient magnitude, about on the order of $\sqrt{\mathbb{E}[\tau]}$ for each prompt x. In particular, more difficult problems with larger expected numbers of attempts will have their policy gradient contributions up-weighted relative to easier problems with smaller expected numbers of attempts.