EVENT-DRIVEN COMPUTE ALLOCATION IN FULLY ASYNCHRONOUS LLM TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) for large language models is computationally intensive, with rollout generation consuming substantial computational resources. Recent fully asynchronous RL systems improve hardware utilization by decoupling rollout generation from model updates, but they leave open a central algorithmic question: **once a rollout job finishes and a generation slot becomes available, how should the system allocate the next unit of compute to maximize learning efficiency?** This compute-allocation question is particularly important in large-scale RL runs, where the goal is to achieve the best possible performance under a fixed compute budget. We propose `EDCA` (**E**vent-**D**riven **C**ompute **A**llocation), a unified framework for prompt sampling and rollout budget allocation in fully asynchronous RL. Unlike existing methods, which typically study either prompt selection or adaptive rollout allocation in synchronous settings, our framework treats *job completion* as the fundamental control point of the learning system. At each completion event, the scheduler chooses between two actions: `continue`, which launches more rollouts for the current prompt, and `replace`, which retires the current prompt and fills the freed slot with a newly sampled one. Across three model backbones and six mathematical reasoning benchmarks, `EDCA` improves average accuracy over fixed-rollout GRPO, including a 2.81% point gain on DeepSeek-R1-Distill-Qwen-1.5B. `EDCA` effective gradient ratio reaches 92.58%, compared with 56.92% for GRPO. When combined with dynamic sampling, `EDCA` reduces rejected samples by 76.1% while completing more policy updates within the same wall-clock budget.

1 INTRODUCTION

Reinforcement learning (RL) has become a central paradigm for post-training large language models (LLMs), enabling a model to improve its policy through feedback collected from its own interactions rather than relying exclusively on static demonstrations (Ouyang et al., 2022; Shao et al., 2024; Guo et al., 2025; Wang et al., 2026b). A particularly successful instantiation is reinforcement learning with verifiable rewards (RLVR), in which an LLM generates candidate trajectories that are evaluated by automatic verifiers, typically producing binary outcome rewards (Li et al., 2025b; Guo et al., 2025; Wang et al., 2026a). These rewards allow the model to reinforce successful reasoning behaviors while suppressing unsuccessful ones. Initially pioneered in mathematical reasoning (OpenAI, 2024; Huang and Yang, 2025), this framework has since been extended to domains like coding and agentic tasks (Team et al., 2026; Xu et al., 2026; Li et al., 2026).

Recent fully asynchronous RL systems use separate GPU pools for streaming rollout generation and policy updates in GRPO pipelines, mitigating long-tail generation stalls and improving GPU utilization (Fu et al., 2026; Han et al., 2025; Zhu et al., 2025). In this setting, improving effective gradient ratio is crucial for training efficiency and has been extensively studied in synchronous RL settings. DAPO (Yu et al., 2026) discards groups in which all rollouts receive the same reward, which incurs substantial additional rollout cost. Similarly, REINFORCE-Ada (Xiong et al., 2025) uses rejection sampling to control the ratio of positive to negative samples. KnapsackRL (Li et al., 2025a) mitigates such waste by reallocating the rollout budget within each batch toward more valuable prompts. However, its synchronous design remains constrained by fixed batch boundaries and cannot fully exploit streaming rollouts. This raises a critical question: **Can we adaptively allocate rollout budgets in asynchronous training without wasting samples?**

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

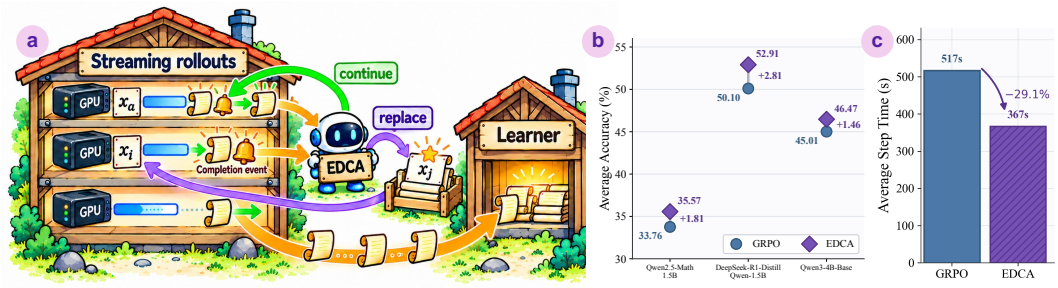


Figure 1: **(a)** In the fully asynchronous framework, EDCA introduces a scheduler that, upon each rollout completion, decides whether to continue sampling the current prompt or replace it with a new one, enabling flexible, event-driven allocation of generation compute. **(b)** EDCA improves average accuracy over GRPO across three model backbones. **(c)** EDCA with dynamic sampling reduces average step time from 517 s to 367 s, saving approximately 29% per step.

In fully asynchronous RL, a rollout worker switches to a new prompt as soon as the current prompt has collected fixed quota of n responses, without waiting for other workers. This motivates us to formulate generation as a $\{\texttt{continue}, \texttt{replace}\}$ decision at each rollout completion event: GRPO selects `replace` after the n-th response. Once a rollout job finishes and a generation slot becomes available, how should the system allocate the next unit of compute to maximize learning efficiency? We propose EDCA, an event-driven framework for compute allocation in fully asynchronous RL, which comprises three components: (i) a **value-guided prompt sampler** that selects new prompts based on estimated learning value; (ii) an **adaptive rollout scheduler** that decides at each prompt completion whether to `continue` sampling the current prompt or `replace` it; and (iii) a **training trigger** that determines when sufficient training data have accumulated to update the policy.

We evaluate EDCA on three model backbones and six mathematical reasoning benchmarks. Compared with GRPO, EDCA improves average accuracy by 2.81% points on DeepSeek-R1-Distill-Qwen-1.5B. We further measure the effective gradient ratio, defined as the fraction of prompt groups in a batch that contain both positive- and negative-reward responses. EDCA achieves a ratio of 92.58%, compared with 56.92% for GRPO. This indicates that more prompt groups provide effective policy-gradient signals, improving the performance gains. Furthermore, EDCA can combine with dynamic sampling, which completes 36.0% more policy update steps within the same wall-clock budget while reducing rejected samples by 76.1%.

Our contributions are summarized as follows:

Contribution

- Building on the characteristics of fully asynchronous RL, we formulate compute allocation as an event-driven decision problem.
- We introduce EDCA which including value-guided prompt sampler and an adaptive `continue`/`replace` scheduler that allocate additional rollouts and training trigger.
- We demonstrate improved mathematical reasoning performance and a higher effective gradient ratio across three model backbones, as well as lower sampling cost when EDCA is combined with dynamic sampling.

2 Preliminary

2.1 Fully asynchronous RL training

Synchronous RL post-training alternates between rollout generation and policy optimization. Each iteration samples a batch of prompts, assigns a rollout budget to each prompt, and waits for all responses before updating the policy. A disaggregated architecture uses separate resources for rollout generation and policy training, allowing the two stages to scale independently (Fu et al., 2026). How-

ever, they may still synchronize at batch boundaries. Fully asynchronous RL removes this shared rollout-batch boundary (Fu et al., 2026; Piché et al., 2025). Prompt sampling, response generation, and policy updates proceed on independent schedules: new prompts are dispatched as rollout capacity becomes available, completed responses are evaluated and buffered, and the trainer updates whenever enough training-ready data have accumulated. Updated weights are transferred to rollout workers while data collection continues. Thus, training batches need not coincide with prompt submission batches, and individual responses need not wait for unrelated rollouts to finish. This overlap reduces idle time caused by variable response lengths.

2.2 Policy gradients and effective signal

Many RLVR methods, including GRPO (Shao et al., 2024), estimate advantages by comparing rewards across responses to the same prompt. For a prompt x_i, let $\{y_{i,j}\}_{j=1}^{N_i}$ denote a group of sampled responses with rewards $\{r_{i,j}\}_{j=1}^{N_i}$. The group mean reward and centered advantages are

$$\bar{r}_i = \frac{1}{N_i}\sum_{j=1}^{N_i} r_{i,j}, \qquad A_{i,j} = r_{i,j} - \bar{r}_i.$$

For responses sampled from π_θ, a simplified policy gradient estimator is

$$\hat{g} = \frac{1}{\sum_{i=1}^{M} N_i}\sum_{i=1}^{M}\sum_{j=1}^{N_i} A_{i,j}\nabla_\theta \log \pi_\theta(y_{i,j} \mid x_i),$$

where M is the number of prompt groups in the update. This expression omits normalization by the group standard deviation, clipping, and KL regularization to isolate the effect of centering.

When all responses within a group receive the same reward, $A_{i,j} = 0$ for every response, and the group's contribution to $\hat{g}$ is zero. With binary rewards $r_{i,j} \in \{0, 1\}$, nonzero centered advantages require both correct and incorrect responses within the group, or equivalently $0 < \bar{r}_i < 1$. Groups containing only correct or only incorrect responses therefore consume rollout compute without contributing to this gradient estimate.

Zero gradient contributions also arise in other policy gradient estimators, but under different conditions. Vanilla REINFORCE without a baseline (Williams, 1992) weights each response by its reward. Under binary rewards, the gradient estimate is zero when all sampled responses are incorrect, but need not be zero when all responses are correct. ReMax (Li et al., 2023) instead uses the reward of a greedily decoded response to the same prompt as its baseline. Its advantage vanishes whenever the sampled response and the greedy response receive the same reward.

3 Related Work

Asynchronous RL systems. Recent work has established fully asynchronous training as a promising systems direction for LLM post-training. Early evidence for asynchronous, off-policy language-model RL appeared in asynchronous RLHF (Noukhovitch et al., 2025). AReaL, LlamaRL, AsyncFlow, and PipelineRL further show that decoupling generation from training can substantially improve throughput and wall-clock efficiency in reasoning-focused LLM RL (Fu et al., 2026; Wu et al., 2025; Han et al., 2025; Piché et al., 2025). More recently, DORA maintains multiple rollout policy versions to accommodate long trajectories while preserving within-trajectory policy consistency and bounding staleness (Hu et al., 2026). These systems primarily address execution efficiency and training stability through scheduling, weight synchronization, and staleness control. Our work complements them by treating rollout allocation as a sequential decision problem over a stream of completed responses. Each completion provides feedback for deciding whether to continue sampling the same prompt or switch to other prompts, allowing allocation to adapt to observed outcomes without waiting for a complete rollout batch or a policy update.

Adaptive rollout allocation. A parallel line of work allocates rollout budgets non-uniformly across prompts. Knapsack RL formulates exploration-budget allocation as a constrained optimization problem under a fixed compute budget (Li et al., 2025a). Reinforce-Ada uses successive elimination and adaptive response allocation for reinforce-style LLM training (Xiong et al., 2025). VIP

allocates rollouts to minimize expected gradient variance under a hard budget constraint (Nguyen et al., 2026). CoBA-RL allocates compute based on an explicit capability-oriented estimate of prompt training value (Yao et al., 2026), while AERO combines adaptive rollout allocation with selective rejection to improve compute efficiency in group-based RL (Zhang et al., 2026). More recently, CERO optimizes rollout allocation across prompts and epochs under a global budget (Zong et al., 2026), and SARA uses sequential stopping to redirect computation from saturated groups to fresh prompts (Voss et al., 2026). These methods are relevant to our setting, but they are usually described from a batch-centric viewpoint. Our paper reinterprets the same family of ideas from the perspective of *completion-event control* in a fully asynchronous system.

4 Towards System and Algorithm Co-Design

In GRPO, when all responses within a rollout group receive the same reward, the group has zero empirical reward variance, causing all group-relative advantages to vanish (Li et al., 2025a; Yu et al., 2026; Chen et al., 2025). DAPO (Yu et al., 2026) filters out zero-variance groups, but the rollouts used to identify and discard such groups still consume computational resources. Knapsack RL (Li et al., 2025a) formulates budget allocation by estimating each prompt's learning value from its empirical success rate. However, these estimates are necessarily noisy and may become stale as the policy evolves, potentially leading to inaccurate value estimates and suboptimal budget assignments. This motivates a natural question: *can rollout budgets be allocated more efficiently and flexibly during rollout generation?*

We find that this challenge can be addressed through system–algorithm co-design in fully asynchronous RL. In particular, streaming rollout generation provides a finer control granularity by **exposing each completion event as a fine-grained control point**. The allocation algorithm can therefore react online whenever a rollout slot becomes available and decide whether to continue the current prompt or assign the slot to another one. We refer to this allocation module as the *scheduler*. As a simple example, consider classical homogeneous budget allocation, where every prompt x_i is assigned the same fixed rollout budget (n). The scheduler uses the completion count as its stopping criterion. Let m_i^e denote the number of completed rollouts for prompt x_i by completion event (e). Although rollout jobs finish asynchronously, the scheduler continues assigning each newly freed slot to the same prompt as long as $m_i^e < n$. Once the (n)-th rollout has completed, it retires x_i and assigns the slot to the next prompt:

$$\pi_e^{\text{fixed}}(s_i^e) = \begin{cases} \texttt{continue}, & m_i^e < n, \\ \texttt{replace}, & m_i^e = n. \end{cases}$$

It is then natural to generalize this deterministic completion rule to an event-driven adaptive policy. At each completion event, the scheduler can inspect the partial outcomes collected for x_i, summarized by the state s_i^e, and choose whether to continue sampling the same prompt or replace it with another one:

$$\pi_e(s_i^e) \in \{\texttt{continue}, \texttt{replace}\}.$$

Under this adaptive algorithm, the rollout budget is no longer fixed in advance but is determined by an outcome-dependent stopping time, $N_i = \tau_i$. This raises a central question: when should the scheduler continue the current prompt or replace it based on partial rollout outcomes?

5 Proposed Approach

In this section, we introduce EDCA, a unified event-driven framework for fully asynchronous RL. The pipeline is shown in Figure 2. EDCA decomposes into three decision modules: (i) **a value-guided prompt sampler**, which assigns newly freed slots to high-value prompts; (ii) **an adaptive rollout scheduler**, which terminates a prompt once its marginal learning value is zero; and (iii) **a training-trigger**, which determines when accumulated resolved data suffices for a policy update. Algorithm 1 summarizes the full pipeline.

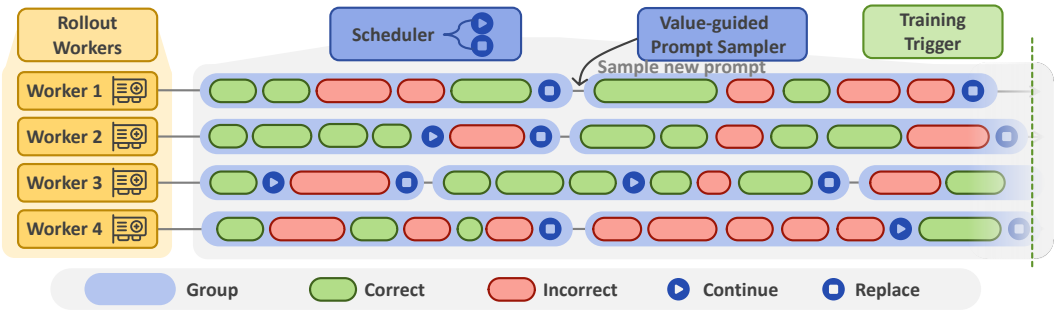


Figure 2: Overview of EDCA. At each prompt completion, the scheduler decides whether to `continue` sampling or `replace` it. Upon replacement, the prompt sampler selects prompt with highest value. Once sufficient samples have been collected, training-trigger initiates policy update.

5.1 Value-Guided Prompt Sampler

We first introduce the Value-Guided Prompt Sampler, which selects a new prompt. Before introducing the Value-Guided Prompt Sampler, we first define the prompt state. For each active prompt x_i, it maintains a prompt state $s_i^e = \left(x_i,\ n_i^e,\ n_{i,+}^e,\ n_{i,-}^e,\ p_i^e,\ \hat{v}_i^e,\ \delta_i^e\right)$, where e indexes rollout completion events, n_i^e is the number of finished rollouts for prompt x_i, $n_{i,+}^e$ and $n_{i,-}^e$ are the numbers of positive and negative outcomes, p_i^e is the empirical success rate, v_i^e is the value of prompt x_i. Inspired by Li et al. (2025a), we define the value v_i^e as the expected improvement in success probability after a policy update:

$$v_i^e \triangleq \mathbb{E}\left[p_i^{e+1} - p_i^e \mid x_i\right], \tag{1}$$

where p_i^e and p_i^{e+1} denote the success probabilities before and after the update, respectively. The exact learning value in Equation 1 is intractable because it requires evaluating the policy after update. Proposition Proposition 1 provides an approximation based on the current success probability.

Proposition 1 (Value approximation (Li et al., 2025a)). *Under a first-order Taylor approximation, the value defined in Equation 1 can be approximated as*

$$v_i^e \approx p_i^e\left(1 - p_i^e\right)^2.$$

In this work, we use the estimated values to prioritize prompt sampling. Specifically, whenever a rollout slot becomes available and the scheduler decides to replace the current prompt, it samples a new prompt with probability proportional to its estimated learning value. For the candidate prompt set $\mathcal{X}^e$, we define the sampling probability q_i^e of prompt x_i as $q_i^e = \frac{w_i^e}{\sum_{x_j \in \mathcal{X}^e} w_j^e}, w_i^e = \max\left(\hat{v}_i^e, \gamma\right)$, where $\gamma > 0$ ensures that every prompt has a nonzero probability of being selected.

5.2 Continuation and Stopping Scheduler

As discussed in section 4, fully asynchronous execution exposes each rollout-completion event as a fine-grained control point. At such an event, the scheduler can decide whether to continue generating rollouts for the currently active prompt or to release the slot and switch to another prompt. For clarity, we divide rollout generation into two stages. Each prompt first receives an initial budget of K rollouts. Whenever a rollout for the active prompt x_i completes at event e, the scheduler updates the prompt state s_i^e and evaluates whether allocating one additional rollout to x_i is preferable to replacing it with another prompt. We define the ideal continuation margin as

$$\Delta_e(s_i^e) = \Gamma_e^{\text{cont}}(s_i^e) - \Gamma_e^{\text{next}},$$

where $\Gamma_e^{\text{cont}}(s_i^e)$ denotes the estimated learning gain per unit compute from continuing the current prompt, and Γ_e^{next} denotes the corresponding gain from switching to the next prompt. Ideally, the scheduler continues the current prompt when $\Delta_e(s_i^e) > 0$ and replaces it otherwise. We define $\Delta_e(s_i^e)$ based on effective-gradient signals. Let $R_i^{1:K_i+c}$ denote the rewards from the initial K_i rollouts and the subsequent c rollouts. We define

$$\Delta_e(s_i^e) = \mathbf{1}\left\{\Pr\left(\widehat{\mathrm{Var}}\left(R_i^{1:K_i+c}\right) > 0 \mid s_i^e\right) > \epsilon\right\} - \mathbf{1}\left\{\widehat{\mathrm{Var}}\left(R_i^{1:K_i}\right) > 0\right\}, \tag{2}$$

where $\widehat{\mathrm{Var}}(\cdot)$ denotes the empirical variance of the corresponding reward, and $\Pr(\cdot \mid s_i^e)$ denotes the probability of the specified event conditioned on the current prompt state s_i^e. The threshold ϵ determines the minimum probability required for continuing to allocate rollouts to the current prompt. Equation 2 can be analyzed in the following two cases:

- When initial K_i responses contain at least one correct response and one incorrect response. The group provides an effective gradient, scheduler reassigns the freed slot to another prompt, so $\Delta_e(s_i^e) = 0$.
- When initial responses receive the same reward. The scheduler considers allocating additional rollouts only when they can obtain a mixed group.

After the initial k_i rollouts, the scheduler selects `replace` if the group already contains both correct and incorrect responses. Otherwise, it considers $c \geq 1$ additional rollouts. We use $\Pr$ denotes $\Pr\left(\widehat{\mathrm{Var}}\left(R_i^{1:k_i+c}\right) > 0 \mid s_i^e\right)$. With binary rewards, all $k_i + c$ responses have nonzero group-relative advantages if the group becomes informative. For additional sampling to be worthwhile, this expectation should exceed c, the number of additional rollouts:

$$(K_i + c)\Pr > c \quad \Longleftrightarrow \quad \Pr > \frac{c}{K_i + c}. \tag{3}$$

Let $c_i^\star = \max\{c \in \mathbb{N}_{>0} : N_{\min} \leq K_i + c \leq N_{\max}\}$ be the largest feasible additional rollout budget. If no such budget exists, the scheduler selects `replace`. Otherwise, it evaluates equation 3 at $c = c_i^\star$, selecting `continue` when the condition holds and `replace` otherwise. We also give two cases for how to calculate $\Pr$ in Appendix A.

Initial Rollout Budget (K_i) To increase the likelihood that the initial rollout group contains both reward outcomes, we adopt a variance-maximizing initialization strategy. For binary rewards, maximizing the empirical variance favors initial groups containing both correct and incorrect responses. We then select the initial rollout count as

$$K_i = \underset{N \in \{N_{\min}, \ldots, N_{\max}\}}{\arg\max} \; 4p_i(N)\left(1 - p_i(N)\right).$$

The lower bound $N_{\min}$ guarantees a minimum number of initial rollouts, while $N_{\max}$ limits the computational cost of initialization.

Algorithm 1 `EDCA`: Event-Driven Compute Allocation

Input: dataset $\mathcal{D}$, sampler q_e, scheduler π_e, training trigger $H(\cdot)$, rollout cap $N_{\max}$

Initialize ready buffer $\mathcal{R} \leftarrow \emptyset$
repeat
  Wait until any rollout job completes
  Identify its prompt x_i and update the state tracker s_i with Equation 3
  if $\pi_e(s_i) =$ `replace` **then**
    Move the resolved group $\mathcal{G}_i$ into $\mathcal{R}$
    Sample a fresh prompt $x' \sim q_e(\cdot)$
    Replace x_i in the working pool by x'
    Launch the first rollout for x'
  else
    Launch c_i rollout for the same prompt x_i
  if $H(\mathcal{R}) = 1$ **then**
    Construct a training batch from $\mathcal{R}$
    Update model parameters
    Asynchronously synchronize the newest policy to rollout workers

Table 1: Performance comparison on six benchmarks. We report mean@16 accuracy (%) and effective gradient ratio. The best result in each column within each backbone is shown in bold. VS denotes value-guided prompt sampler. EG Ratio denotes effective gradient ratio.

Method	AIME 24	AIME 25	AMC	MATH	Minerva	Olympiad	Avg.	EG Ratio
Qwen2.5-Math-1.5B								
GRPO	13.54	10.63	49.77	71.31	22.50	34.81	33.76	41.41
Reinforce-Ada	15.63	6.88	47.14	70.00	21.88	34.94	32.74	32.81
EDCA w/o VS	**16.88**	10.42	**52.41**	71.31	25.31	36.13	35.41	68.65
EDCA	16.25	**11.67**	50.75	**72.44**	**26.00**	**36.31**	**35.57**	**85.06**
DeepSeek-R1-Distill-Qwen-1.5B								
GRPO	34.79	27.71	71.08	**88.56**	29.94	48.50	50.10	56.92
Reinforce-Ada	35.21	26.25	68.75	86.69	28.44	46.06	48.57	38.67
EDCA w/o VS	40.00	28.33	72.36	86.68	30.88	**50.00**	51.38	73.24
EDCA	**43.96**	**29.38**	**75.45**	88.37	**31.69**	48.62	**52.91**	**92.58**
Qwen3-4B-Base								
GRPO	26.46	21.04	58.73	82.75	35.33	45.75	45.01	53.13
Reinforce-Ada	20.63	**24.17**	56.55	80.69	33.13	45.69	43.47	39.84
EDCA w/o VS	**27.92**	22.71	58.13	**84.56**	35.44	**48.75**	46.25	58.33
EDCA	27.50	21.46	**60.69**	**84.87**	**37.06**	47.25	**46.47**	**65.23**

5.3 Training Trigger and Batch Construction

The third decision module determines when the trainer should launch a policy update. Let $\mathcal{R}_e$ denote the ready buffer at event e, containing resolved prompt groups or completed trajectories that have already exited the working pool. A training trigger is a function

$$H(\mathcal{R}_e) \in \{0, 1\},$$

which fires when enough data has accumulated.

Typical choices include:

$$\texttt{fixed_prompt}: \quad H(\mathcal{R}_e) = \mathbb{I}\big[|\mathcal{R}_e^{\text{prompt}}| \geq B_p\big],$$
$$\texttt{fixed_trajectory}: \quad H(\mathcal{R}_e) = \mathbb{I}\big[|\mathcal{R}_e^{\text{traj}}| \geq B_\tau\big].$$

Once $\mathcal{R}_e$ reaches a fixed number of samples, we trigger the trainer to construct a batch and apply the base RL update, thereby keeping the batch size constant across updates and ensuring a fair comparison among prompts with different rollout counts.

6 Experiments

6.1 Experiment Setting.

We implement `EDCA` for all methods using the large-scale reinforcement learning framework VeRL (Sheng et al., 2024). We construct the training sets from DAPO-Math-17K (Yu et al., 2026), which contains approximately 17K mathematical reasoning prompts with ground-truth answers for rule-based verification. We evaluate `EDCA` on three models: `Qwen2.5-Math-1.5B` (Yang et al., 2024), `DeepSeek-R1-Distill-Qwen-1.5B` (Guo et al., 2025), and `Qwen3-4B-Base` (Yang et al., 2025). The training sets are augmented with offline pass-rate estimates computed from eight sampled responses and prompts for which all eight responses are correct are filtered out.

We use a batch size of $B = 128$ with $N = 8$ responses per prompt. The actual number of responses is adaptively selected from $[4, 14]$. All models are trained for 500 optimization steps. Maximum response lengths are 16,384 tokens for training and 32,768 for evaluation, except for Qwen2.5-Math-1.5B where both are set to 2,048 tokens. We employ fixed trajectory tragger training, token-mean

loss aggregation, and on-policy updates with a staleness threshold of -1. For evaluation, we follow Luo et al. (2025) and consider six mathematical reasoning benchmarks: AIME24, AIME25, AMC, MATH, MINERVA, and OLYMPIAD Bench (OLYMPIAD for short). All results are averaged over 16 generated responses per problem. We compare EDCA with two baselines, GRPO and Reinforce-Ada (Pos) (Xiong et al., 2025). Details of both baselines are provided in Appendix E.

6.2 MAIN RESULTS

Both EDCA and EDCA w/o VS outperform the non-EDCA baselines on most benchmarks and backbones. For Qwen2.5-Math-1.5B, EDCA improves the average accuracy over the strongest baseline GRPO by 1.81 points, and for DeepSeek-R1-Distill-Qwen-1.5B by 2.81 points, including a gain of 9.17 points on AIME 24. On Qwen3-4B-Base, EDCA achieves the best average of 46.47%, with gains of 2.12 points on MATH.

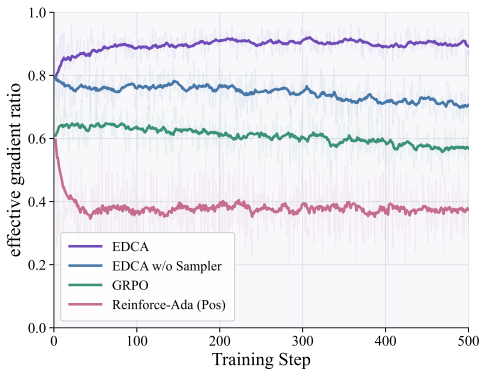


Figure 3: Effective gradient ratio during training. EDCA consistently maintains the highest effective gradient ratio.

To examine whether EDCA produces more informative rollout groups, we measure the effective-gradient ratio at training step 500. Taking DeepSeek-R1-Distill-Qwen-1.5B as an example, we examine whether EDCA increases the effective-gradient ratio. EDCA reaches 92.58%, compared with 73.24% for EDCA w/o Value, 56.92% for GRPO, and 38.67% for REINFORCE-Ada (Pos). We also report the effective gradient in training process. As shown in Figure 3, EDCA maintains a higher ratio than the baselines throughout training. These results indicate that EDCA allocates the rollout budget toward prompts that provide effective policy-gradient signals.

6.3 COMBINING EDCA WITH DYNAMIC SAMPLING

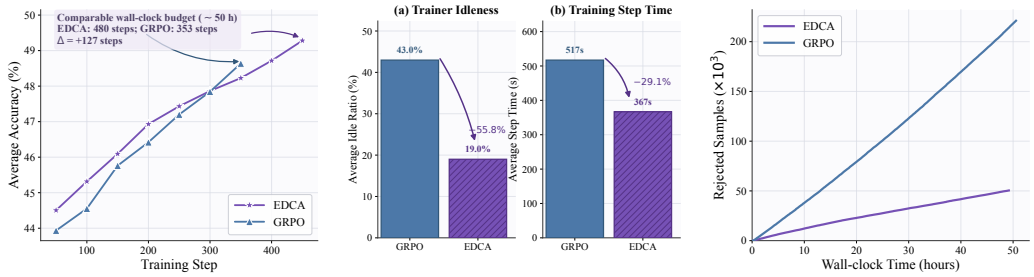
We further combine EDCA with dynamic sampling under rejection sampling on DeepSeek-R1-Distill-Qwen-1.5B . Dynamic sampling repeatedly generates trajectories until a sufficient number of valid samples has been collected. Although this strategy improves batch quality, uniform rollout allocation can waste substantial computation on prompts whose trajectories are repeatedly rejected. EDCA complements dynamic sampling by allocating exploration budgets according to prompt-level utility, reducing redundant generation while retaining competitive policy performance.

As shown in Figure 4, under the same wall-clock budget of approximately 46.0 hours, EDCA completes 480 training steps, compared with only approximately 353 steps for GRPO. This represents 127 additional updates, or a 36.0% increase in completed training steps. The gain is driven by substantially improved system efficiency: EDCA reduces the average step time from 517.1 to 366.7 seconds (29.1% less time per step)[1], and lowers the trainer idle ratio from 43.0% to 19.0%. Moreover, within the same time budget, the number of rejected samples decreases from 197,912 to 47,290, reducing the number of rejected samples by 150,622 (76.1%). More analysis is shown in Appendix D. These results show that EDCA complements dynamic sampling by directing rollouts toward prompts more likely to yield useful trajectories, improving both sampling efficiency and training throughput.

6.4 UNDERSTANDING EDCA

To understand where the gains of EDCA come from, we ablate its components on Qwen2.5-Math-1.5B. As shown in Figure 5, the full EDCA achieves an average accuracy of 35.57%. Removing the value-guided prompt sampler and selecting the next prompt sequentially whenever `replace` is triggered reduces accuracy to 35.41%. Further removing the adaptive rollout scheduler, so that each prompt is replaced after its initial K rollouts, yields 34.87%. Finally, removing the adaptive initial-

[1] Average step times are computed over 480 training steps for EDCA and 353 training steps for GRPO.



(a) Training performance. (b) System efficiency. (c) Rejected samples.

Figure 4: Comparison between GRPO and EDCA with dynamic sampling. EDCA maintains competitive performance while reducing the trainer idle ratio, time, and number of rejected samples.

ization of K recovers GRPO, which achieves 33.76%. These results show that prompt sampling, rollout scheduling, and budget initialization each contribute to the overall gain of 1.81% points.

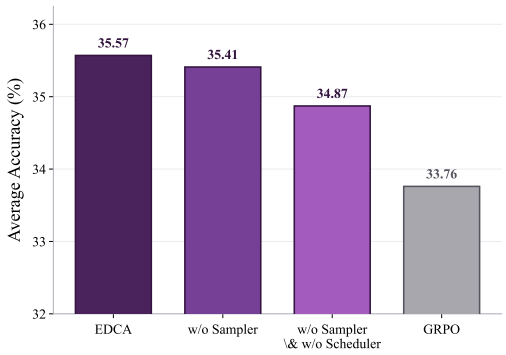


Figure 5: Ablation of the value-guided prompt sampler, adaptive rollout scheduler, and initialization of K in EDCA on Qwen2.5-Math-1.5B. Components are removed cumulatively. Average accuracy decreases as components are removed, indicating that each contributes to EDCA's performance.

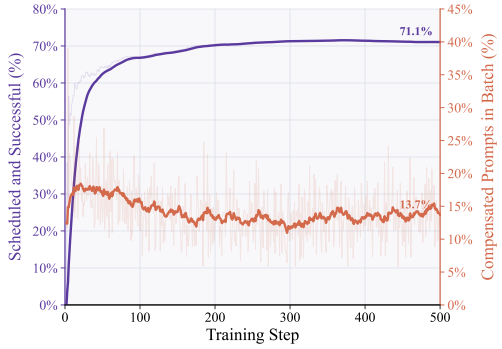


Figure 6: Behavior of the compensation scheduler during training. The two axes show the success rate of scheduled compensation and the fraction of prompts in each batch selected for compensation. On average, approximately 13.7% of prompts are selected, and compensation succeeds for 71.1% of them.

Second, we examine the behavior of the scheduler, including how frequently prompts are scheduled and how often the scheduled compensation succeeds as shown in Figure 6. Approximately 13.7% of the prompts in each batch are scheduled, while 71.1% of the scheduled compensation attempts succeed. This indicates that EDCA can allocate additional rollouts to prompts which successfully produce the desired outcome variation.

7 Conclusion

This work studied how to allocate rollout compute efficiently in fully asynchronous RL for large language models. In particular, we focused on the decision made when a rollout completes and a generation slot becomes available: whether to continue sampling the current prompt or switch to a new one. We introduced EDCA, an event-driven framework for compute allocation in fully asynchronous RL. At rollout completion events, it combines value-guided prompt sampling with an adaptive `continue`/`replace` scheduler to direct generation toward prompts with greater potential training value. Across three model backbones and six mathematical reasoning benchmarks, EDCA improves average accuracy and increases the fraction of prompt groups that provide effective gradients. When combined with dynamic sampling, it also reduces rejected samples and average step time. These results show that EDCA can improve both sampling efficiency and training throughput.

AI USE STATEMENT

Large language model are used to assist with language editing, manuscript organization, and LaTeX formatting. The authors reviewed and revised all AI-assisted text for technical accuracy and consistency with the research. The authors take full responsibility for the methodology, analyses, experimental results, and conclusions in the final manuscript.

REPRODUCIBILITY STATEMENT

We provide code for reproduction and describe the fully asynchronous training setup and the three components of `EDCA`: value-guided prompt sampling, adaptive `continue`/`replace` scheduling, and the training trigger. We also report the model, datasets, reward verification and evaluation procedures, rollout budgets, and training hyperparameters used in our experiments. The appendix provides further implementation details and the derivation of the continuation criterion.

REFERENCES

Peter Chen, Xiaopeng Li, Ziniu Li, Xi Chen, and Tianyi Lin. Spectral policy optimization: Coloring your incorrect reasoning in grpo. *arXiv preprint arXiv:2505.11595*, 2025.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *Advances in Neural Information Processing Systems*, 38:36256–36282, 2026.

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

Zhenyu Han, Ansheng You, Haibo Wang, Kui Luo, Guang Yang, Wenqi Shi, Menglong Chen, Sicheng Zhang, Zeshun Lan, Chunshi Deng, et al. Asyncflow: An asynchronous streaming rl framework for efficient llm post-training. *arXiv preprint arXiv:2507.01663*, 2025.

Tianhao Hu, Xiangcheng Liu, Yuchun Miao, Youshao Xiao, Hongyu Zang, Yang Zheng, Xuan Huang, Jinrui Ding, Yufei Zhang, Yu Yang, et al. Dora: A scalable asynchronous reinforcement learning system for language model training. *arXiv preprint arXiv:2604.26256*, 2026.

Yichen Huang and Lin F Yang. Gemini 2.5 pro capable of winning gold at imo 2025. *arXiv preprint arXiv:2507.15855*, 2025.

Chengpeng Li, Zhengyang Tang, Ziniu Li, Mingfeng Xue, Keqin Bao, Tian Ding, Ruoyu Sun, Benyou Wang, Xiang Wang, Junyang Lin, et al. Teaching language models to reason with tools. *Advances in Neural Information Processing Systems*, 38:66843–66892, 2026.

Ziniu Li, Tian Xu, Yushun Zhang, Zhihang Lin, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. Remax: A simple, effective, and efficient reinforcement learning method for aligning large language models. *arXiv preprint arXiv:2310.10505*, 2023.

Ziniu Li, Congliang Chen, Tianyun Yang, Tian Ding, Ruoyu Sun, Ge Zhang, Wenhao Huang, and Zhi-Quan Luo. Knapsack rl: Unlocking exploration of llms via optimizing budget allocation. *arXiv preprint arXiv:2509.25849*, 2025a.

Ziniu Li, Pengyuan Wang, Tian Xu, Tian Ding, Ruoyu Sun, and Yang Yu. Review of reinforcement learning for large language models: Formulations, algorithms, and opportunities, 2025b.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl, 2025. Notion Blog.

Trung Hieu Nguyen, Bao Nguyen, Wenao Ma, Yuzhi Zhao, Ruifeng She, and Viet Anh Nguyen. Adaptive rollout allocation for online reinforcement learning with verifiable rewards. In *International Conference on Learning Representations*, 2026.

Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron C. Courville. Asynchronous RLHF: faster and more efficient off-policy RL for language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore*. OpenReview.net, 2025.

OpenAI. Learning to reason with LLMs. OpenAI Blog, Sep 2024. https://openai.com/index/learning-to-reason-with-llms/.

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

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. Pipelinerl: Faster on-policy reinforcement learning for long sequence generation. *arXiv preprint arXiv:2509.19128*, 2025.

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

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

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Elena Voss, Marcus Hale, and Sofia Reyes. Early verdicts, better budgets: Sequential adaptive rollout allocation for compute-efficient rlvr. *arXiv preprint arXiv:2607.26253*, 2026.

Peng-Yuan Wang, Ziniu Li, Tian Xu, Bohan Yang, Tian-Shuo Liu, ChenYang Wang, Xiong-Hui Chen, Yi-Chen Li, Tianyun Yang, Congliang Chen, et al. Off-policy value-based reinforcement learning for large language models. *arXiv preprint arXiv:2603.23355*, 2026a.

Peng-Yuan Wang, Tian-Shuo Liu, Chenyang Wang, Ziniu Li, Yidi Wang, Shu Yan, Chengxing Jia, Xu-Hui Liu, Xinwei Chen, Jiacheng Xu, et al. A survey on large language models for mathematical reasoning. *ACM Computing Surveys*, 58:1–35, 2026b.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

Bo Wu, Sid Wang, Yunhao Tang, Jia Ding, Eryk Helenowski, Liang Tan, Tengyu Xu, Tushar Gowda, Zhengxing Chen, Chen Zhu, et al. Llamarl: A distributed asynchronous reinforcement learning framework for efficient large-scale llm training. *arXiv preprint arXiv:2505.24034*, 2025.

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. *arXiv preprint arXiv:2510.04996*, 2025.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhiyuan Yao, Yi-Kai Zhang, Yuxin Chen, Yueqing Sun, Zishan Xu, Yu Yang, Tianhao Hu, Qi Gu, Hui Su, and Xunliang Cai. Coba-rl: Capability-oriented budget allocation for reinforcement learning in llms. *arXiv preprint arXiv:2602.03048*, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Zhi Zhang, Zhen Han, Costas Mavromatis, Qi Zhu, Yunyi Zhang, Sheng Guan, Dingmin Wang, Xiong Zhou, Shuai Wang, Soji Adeshina, et al. Train less, learn more: Adaptive efficient rollout optimization for group-based reinforcement learning. *arXiv preprint arXiv:2602.14338*, 2026.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling, 2025. GitHub repository.

Yiming Zong, Yige Wang, and Jiashuo Jiang. Cross-epoch adaptive rollout optimization for rl post-training. *arXiv preprint arXiv:2606.05606*, 2026.

A Probability of Obtaining an Effective Gradient

Let p_i denote the estimated probability that a rollout for prompt x_i is correct. Assuming that future rollout rewards are conditionally independent given p_i, we compute $s_i(c)$ according to the rewards of the initial k_i responses. If they are all incorrect, the group remains uninformative only when all c additional responses are also incorrect, an event with probability $(1-p_i)^c$. If they are all correct, the corresponding probability is p_i^c. Hence,

$$\Pr\left(\widehat{\mathrm{Var}}\left(R_i^{1:k_i+c}\right) > 0 \mid s_i^e\right) = \begin{cases} 1-(1-p_i)^c, & \text{if all initial responses are incorrect,} \\ 1-p_i^c, & \text{if all initial responses are correct.} \end{cases}$$

B Training Hyperparameters

Training data and rollout configuration. We train all models on DAPO-Math-17K, where each problem is paired with a ground-truth answer for rule-based verification. The maximum prompt length is 2,048 tokens. Fixed-rollout baselines generate $N = 8$ responses per prompt, while Knapsack-GRPO uses $N = 8$ as its nominal rollout budget and dynamically adjusts the actual number of responses. We use a prompt mini-batch size of 128.

The maximum training and evaluation response lengths are 16,384 and 30,720 tokens for DPSK-R1-Distill-1.5B. For Qwen2.5-Math-1.5B and Qwen3-4B-Base, both limits are 2,048 tokens. During training, responses are sampled with temperature 1.0, top-$p = 1.0$, and no top-k truncation. Evaluation uses temperature 0.6 and 16 responses per problem.

Optimization. We optimize the policy using AdamW with a constant learning rate of 2×10^{-6}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, zero weight decay, and gradient-norm clipping at 1.0. No learning-rate warmup is used. We perform one PPO epoch per rollout batch and use asymmetric clipping with policy-ratio bounds $[0.8, 1.28]$. The policy loss is averaged over valid response tokens. Both the KL and entropy regularization coefficients are set to zero. Parameters are synchronized after every update, yielding fully on-policy training with a staleness threshold of -1.

Distributed training. Training uses bfloat16 precision, gradient checkpointing, FlashAttention-2, and FSDP2 without parameter or optimizer offloading. Rollout generation is implemented with vLLM using tensor parallelism of one, chunked prefill, and a GPU-memory-utilization target of 0.85. Qwen2.5-Math-1.5B uses four rollout and four training GPUs. DPSK-R1-Distill-1.5B and Qwen3-4B-Base each use eight rollout and eight training GPUs.

C Prompt Templates

C.1 Prompts for Training and Inference

We use the same prompt format for on-policy training and evaluation. Each problem is followed by an instruction requesting explicit reasoning and a boxed final answer:

```
{problem}
Please reason step by step, and put your final answer
within \boxed{}.
```

The resulting message is formatted using the chat template associated with each model checkpoint. For Qwen3-4B-Base, thinking mode is explicitly disabled; the other models use their native chat-template behavior. Generation terminates at the tokenizer's end-of-sequence token, and correctness is determined by extracting and verifying the final boxed answer.

D Additional Efficiency Analysis

We further examine EDCA from the perspectives of sample use and system efficiency. Figure 7 shows performance relationship with generated samples, training step time, and idle ratio. These

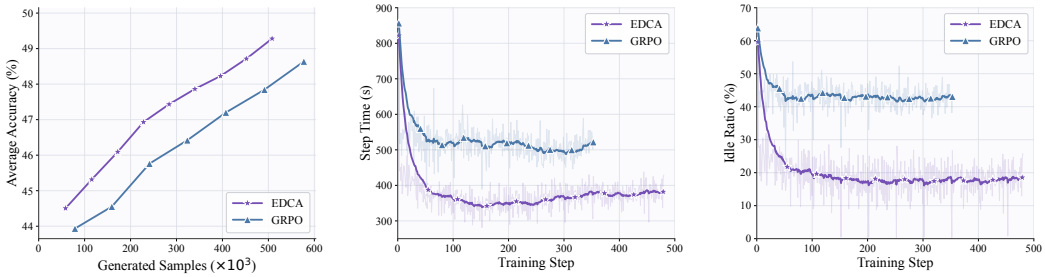


Figure 7: **Additional efficiency analysis. (a)** Performance versus the number of generated samples. **(b)** Training step time. **(c)** Idle ratio.

measurements complement the accuracy results by showing how training progress relates to sampling cost and execution efficiency.

E BASELINE IMPLEMENTATIONS.

For the GRPO baseline, we generate a fixed group of $N = 8$ responses for each prompt and compute advantages by normalizing rewards within each group. For Reinforce-Ada (Pos), we implement positive-triggered adaptive sampling. Responses are generated in chunks of two, and generation stops once at least one correct response has been observed, subject to a maximum of $N_{\max} = 14$:

$$N_i = \min \left\{ n \in \{2, 4, \ldots, 14\} : \sum_{j=1}^{n} r_{i,j} \geq 1 \right\},$$

with $N_i = 14$ if no correct response is obtained. For a controlled comparison, all methods use the same GRPO training objective and a fixed total number of generated responses per update; they differ only in how the rollout budget is allocated across prompts.