TRUST THE CRITIC MORE

Anonymous authors
Paper under double-blind review

ABSTRACT

Standard language model RL algorithms credit every token of a long rollout with the same advantage determined by the terminal reward. Actor–critic methods can provide finer-grained credit assignment, but learned critics are generally considered too inaccurate to trust for RL for LLMs. In recent works, even when a critic is present, it is used only for baseline estimation, so every trajectory must be rolled out to its terminal reward (Venkatraman et al., 2026; Pan et al., 2026). We introduce Actor-Critic with Action Chunking (AC2) that removes the need to roll every trajectory to completion. AC2 instead assigns credit to action chunks: short continuations of prefixes of past trajectories. A learned critic scores the state reached at the end of each action chunk, allowing the policy to update without observing a terminal reward. We make critic-based credit assignment reliable through three design choices. First, we introduce *local readiness* which uses critic-based updates on a problem only when the critic is sufficiently accurate on that particular problem. Second, when available, we provide the critic with a reference solution from a previous successful rollout. Third, we assign credit over action chunks of 10k tokens rather than individual tokens, giving the critic a more meaningful amount of work to evaluate. We train Qwen3-4B on FineProofs-RL using AC2 and evaluate on IMO-ProofBench. AC2 exceeds GRPO's peak validation score of 18.50% using 2.5× fewer decoding FLOPs. This gain comes from two sources, (1) AC2 requires 25% fewer training steps to reach this score, and (2) each step generates fewer tokens because the policy does not need to continue every trajectory to completion. Conceptually, we demonstrate that we can remove the need to rollout every trajectory to completion, opening up a large previously unexplored design space for LLM RL algorithms.

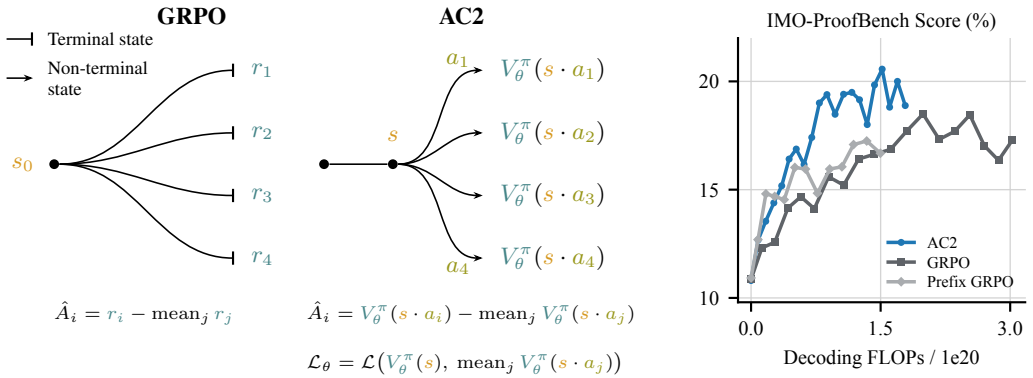

Figure 1: Advantage estimation in GRPO and AC2. GRPO rolls every sample out from the root to a terminal reward. AC2 starts from a replayed prefix s, generates a short chunk a_i per sample, and reads the critic at the chunk's end, $V_\theta^\pi(s \cdot a_i)$, where $\cdot$ denotes concatenation. The same group also trains the critic: $\mathcal{L}$ is a classification loss that fits the critic's prediction at the prefix to the group-mean value, its bootstrapped target (section 3.3). Right: mean score against decoding compute for AC2, GRPO, and Prefix GRPO, which applies GRPO to full-length continuations of replayed prefixes (section 4.1).

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.

1 INTRODUCTION

Standard reinforcement learning algorithms today, such as Group Relative Policy Optimization (GRPO) (Shao et al., 2024), treat the entire rollout as a single action and credit every token of a long rollout with the same advantage that depends solely on the terminal reward. This provides only coarse credit assignment. For example, a minor calculation error at the end of a math proof can cause an otherwise correct and valuable reasoning trajectory to be penalized.

If we instead view a rollout as a sequence of many small actions, actor–critic methods can assign credit to each action individually, providing dense learning signal throughout the trajectory. However, current LLM reinforcement learning methods largely avoids using critics for fine-grained credit assignment, and at most use them as baselines while still relying on terminal rewards for advanatage estimation. In essence, learned critics are considered too inaccurate to be used in isolation without any terminal reward (Kazemnejad et al., 2025).

We revisit this reliance on terminal rewards by introducing Actor-Critic with Action Chunking (AC2), which instead assigns credit to *action chunks*: short continuations of prefixes of past trajectories. A learned critic scores the state reached at the end of each action chunk, allowing the policy to update without rolling out to a terminal reward (Figure 1). Formally, let π_θ and V_θ^π be the policy and value function. From a partial chain-of-thought prefix s, we sample g short token chunks a_i of the same length. Each continuation is scored by the critic at its endpoint, $v_i = V_\theta^\pi(s \cdot a_i)$, where $\cdot$ denotes concatenation, and receives advantage $\hat{A}_i = v_i - \frac{1}{g}\sum_j v_j$. No terminal reward is required for these updates.

Removing the need for a terminal reward on every update opens up a part of the design space for LLM reinforcement learning that has not been explored. AC2 benefits from this in two ways. Firstly, each step can generate far fewer tokens by only rolling out groups of action-chunks instead of full trajectories. Secondly, the actor receives finer-grained supervision of a single advantage for each action-chunk as opposed to trajectory.

We identify three components of the algorithm that make fully critic based advantage estimates reliable. First, we introduce *local readiness*, which measures the critic's error on the particular problem the actor is rolling out. Before the critic becomes sufficiently accurate on a problem, we generate a group of continuations to completion from a sampled prefix. We use their terminal rewards as endpoint values for policy updates and fit the critic to the average terminal reward. After the problem meets the readiness criteria, we instead generate action chunks, use the critic to assign endpoint values for policy updates, and update the critic using the average predicted value. Second, when available, we provide reference solutions from previous successful rollouts on the problem to the critic in context. Third, we use an action-chunk budget of 10k tokens with a full rollout budget of 50k tokens, providing a more meaningful unit of credit assignment than a single token.

We train Qwen3-4B-Thinking-2507 Yang et al. (2025) on FineProofs-RL (LM-Provers et al., 2026) and evaluate on IMO-ProofBench (Luong et al., 2025), a collection of math olympiad level problems. AC2 exceeds GRPO's peak mean score of 18.50% using 2.5× fewer decoding FLOPs (Figure 1). This gain comes from two reasons: (1) AC2 requires 25% fewer training steps: it surpasses this score at step 90, while GRPO takes 120 steps to reach it; and (2) AC2 generates fewer tokens per update because it does not need to continue every trajectory to completion. AC2 also achieves a higher peak mean score of 20.57%, compared with 18.50% for GRPO. Decoding FLOPs are highly correlated with GPU hours in our main RL run (Appendix A.5). We use them as a normalized compute proxy throughout the paper, since different experiments use different numbers of GPUs.

We provide evidence that all three components: local readiness, providing reference solution to critic, and action-chunking are all integral to the success of the agorithm. We find that runs without local readiness plateau early because they apply an imprecise critic to some problems (Figure 3 right). When we reduce action-chunk size from 10k to 2k tokens, performance plateaus earlier than the 10k run (Figure 3 right). Finally, we show the critic error is lower when provided with a correct reference solution in context (Figure 2, fourth panel). We also demonstrate two variants of the AC2 algorithm that perform comparably, the first samples a single action-chunk for each critic-ready problem, using $\hat{A} = V_\theta^\pi(s \cdot c_i) - V_\theta^\pi(s)$. The second uses a highly stale replay buffer for sampling trajectory prefixes (Figure 3 left). We present these variants because they broaden the space of RL algorithms, for example, by enabling algorithms that leverage existing trajectories more extensively.

Conceptually, our results can be summarized as showing that LLM RL can trust a learned critic far more than current methods do. Once the critic is accurate, rolling every trajectory out to a terminal reward is no longer required, and removing that requirement opens up the design space. AC2 is one example of what this makes possible. It assigns credit to individual action chunks, generates a small number of tokens per step, and learns from off-policy initial states drawn from a replay buffer.

2 PRELIMINARIES

We consider the problem of reinforcement learning with verifiable rewards. Let $\mathcal{D}$ be the question distribution and $\mathcal{V}$ the model vocabulary. A policy π_θ with parameters θ maps a question $x \sim \mathcal{D}$ to an autoregressively generated response of length T tokens, $y = (y_1, \ldots, y_T)$, $y_t \in \mathcal{V}$, which a deterministic verifier scores $R(x, y) \in [0, 1]$. The learning objective is the expected verifier score $J(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\,\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}\big[\,R(x,y)\,\big]$. R is only defined only on complete responses.

Token-level MDP. We will consider the following MDP. Let $s_t = x \cdot y_{<t}$ where $\cdot$ denotes string concatenation, with $s_0 = x$ and $a_t = y_t \in \mathcal{V}$. The transition dynamics are simply $s_{t+1} = s_t \cdot a_t$. The reward of the MDP is $r_t = 0$ for $t < T$ and $r_T = R(x, y)$. Let $V^\pi(s)$ denote the expected terminal verifier score when following π from prefix s (equal to $R(x, y)$ for a complete response) and $Q^\pi(s, a)$ the expected score after taking a at s and following π. Since transitions are deterministic and rewards are terminal-only we have

$$Q^\pi(s,a) \;=\; V^\pi(s\cdot a), \quad A^\pi(s,a) \;=\; Q^\pi(s,a) - V^\pi(s) \;=\; V^\pi(s\cdot a) - V^\pi(s). \tag{1}$$

One can also reform the MDP to have a denote a fixed-size chunk of consecutive tokens, and these identities are unchanged.

Policy gradients and advantage estimation. The policy gradient theorem (Sutton et al., 1999) states that, for any action-independent baseline $b(s)$ and policy π_θ with state visitation distribution d^π,

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{s\sim d^\pi,\, a\sim\pi_\theta}\Big[\,\nabla_\theta \log \pi_\theta(a \mid s)\,\big(Q^\pi(s,a) - b(s)\big)\Big]. \tag{2}$$

Note that the reward to go does not appear in equation 2. An algorithm needs some estimate $\hat{A}_t$ of $A^\pi(s_t, a_t)$, and the choice of estimator is what separates existing LLM policy gradient methods. Let V be an approximate state-value function, set to $R(x, y)$ on a complete response, and let $\lambda \in [0, 1]$ control the decay of residual weights in generalized advantage estimation (GAE, Schulman et al. (2018)). With the one-step residual $\delta_t^V = V(s_{t+1}) - V(s_t)$,

$$\hat{A}_t^{\mathrm{GAE}(\lambda)} \;=\; \sum_{l=0}^{T-t} \lambda^l\, \delta_{t+l}^V. \tag{3}$$

In the case when $V \neq V^\pi$, for example in practice when we use a learnt value model, low λ corresponds to low variance and high bias estimators, and high λ the converse. Importantly, once $\lambda > 0$, estimating $\hat{A}_t^{\mathrm{GAE}(\lambda)}$ requiers evaluating the terminal reward. The two λ endpoints take particularly simple forms:

$$\hat{A}_t^{\mathrm{GAE}(0)} = V(s_t \cdot a_t) - V(s_t), \qquad \hat{A}_t^{\mathrm{GAE}(1)} = R(x,y) - V(s_t). \tag{4}$$

The first is the single residual δ_t^V, the second follows by telescoping the residuals through the complete response. At $\lambda = 0$, advantages at nonterminal transitions are estimated entirely using the learned value function, without the terminal reward. At $\lambda = 1$ it is the realized verifier score, with the value function serving *only as a baseline*.

Existing RLVR methods estimate the advantage at $\lambda \approx 1$. Modern RLVR pipelines optimize $J(\theta)$ with a PPO-style clipped surrogate (Schulman et al., 2017). Let θ_{old} be the fixed parameters of the policy that generated the sampled responses and $\rho_t(\theta) = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)$ denote the off-policy importance sampling ratio. With lower and upper clipping parameters ϵ_{low} and ϵ_{high}, the PPO loss is

$$\mathcal{L}(\theta) \;=\; -\,\mathbb{E}_{x\sim\mathcal{D},\; y\sim\pi_{\theta_{\mathrm{old}}}(\cdot|x)}\Big[\,\textstyle\sum_t \min\big(\rho_t(\theta)\,\hat{A}_t,\; \mathrm{clip}(\rho_t(\theta), 1-\epsilon_{\mathrm{low}}, 1+\epsilon_{\mathrm{high}})\,\hat{A}_t\big)\Big]. \tag{5}$$

Most RLVR methods differ mainly in the estimator $\hat{A}_t$. All of them use λ (equation 3) very close to 1. For comparison, PPO-based RLHF (Ouyang et al., 2022) trains a token-level value network V_ϕ and uses $\hat{A}_t$ from equation 3 with $\lambda \approx 0.95$. In RLVR, GRPO (Shao et al., 2024) replaces the learned value network with a baseline computed from a group of sampled responses. For each question x, it samples G responses $y_1, \ldots, y_G \sim \pi_{\theta_{\text{old}}}(\cdot \mid x)$ and computes their mean reward, $V = \frac{1}{G}\sum_{j=1}^{G} R(x, y_j)$. Every token in response y_i receives the same advantage, $\hat{A}_{i,t} = R(x, y_i) - V$, optionally divided by the group's reward standard deviation. The group mean is a Monte Carlo estimate of the root-state value $V^\pi(s_0)$, so GRPO uses the $\lambda = 1$ estimator with a shared baseline.

Other RLVR methods learn a value model but use its predictions only as a baseline, retaining the $\lambda = 1$ advantage estimator. Like GRPO, these methods estimate advantages from the realized terminal reward but they differ in how they construct the baseline (more details in Section 5). Our method differs from the previous works by choosing $\lambda = 0$, using the critic to remove the dependency on terminal rewards for some problems.

3 Actor-Critic with Action Chunking

We present Actor-Critic with Action Chunking (AC2). Let $\mathcal{D}$ be a set of training problems that admit a terminal reward. AC2 trains a policy π_θ and value model V_θ^π that share parameters θ. Our method is motivated by a single guiding principle. *When we know the value function is accurate, we use it to update the policy*. Each step of the AC2 proceeds as follows (see Algorithm 1):

1. **Problem sampling.** We maintain a replay buffer $\mathcal{B}$ of complete rollouts on prior problems. At each step we sample n_{refill} new problems from $\mathcal{D}$ and n_{batch} complete trajectories from $\mathcal{B}$.
2. **Policy sampling.** For each sampled new problem, we do a rollout to the end, and use these to refill the buffer for the next step. For each buffer sampled trajectory, we cut to a random prefix of tokens s_{prefix}, and rollout a group number g of tokens from there. If our value model is deemed accurate on the problem, which we call *ready* (section 3.1), then we rollout a chunk of b tokens. If not then we rollout until the end of the trajectory. Let c_i denote these sampled tokens for the ith entry in the group. This gives us a group of visited states $\{s \cdot c_i\}_{i=1}^{g}$
3. **Advantage estimation.** To calculate the advantage we need the expected return from each continuations state $s \cdot c_i$. If $s \cdot c_i$ is a complete trajectory this is simply the terminal reward r_i. Otherwise we use the critic's estimate $V_\theta^\pi(s \cdot c_i)$. Writing v_i for endpoint value, the advantage is $\hat{A}_i = v_i - \text{mean}_j(v_j)$.
4. **Actor update.** We take a clipped policy gradient step with $\hat{A}_i$ on the newly sampled tokens, leaving the replayed prefix as conditioning context that receives no loss.
5. **Critic update.** We fit V_θ^π to each prefix's group-mean value $\text{mean}_i(v_i)$. If V_θ^π was not ready on the problem, then this target will be the average terminal reward over the group. If it was ready, the target will be an average over the value model's own estimates $\text{mean}_j\big(V_\theta^\pi(s \cdot c_j)\big)$.

3.1 Problem and policy sampling

We begin by sampling states to generate actions for. We keep a replay buffer $\mathcal{B}$ of complete rollouts (that are not necessarily correct) on prior problems. We sample n_{refill} fresh problems from $\mathcal{D}$ without replacement and n_{batch} stored trajectories from $\mathcal{B}$. For each of these n_{batch} trajectories we choose a random cut position after the problem statement. We keep all tokens before the cut. We refer to this state as s_{prefix}. Let $\mathcal{S}_{\text{refill}}$ be the set of n_{refill} fresh states, each a problem statement with an empty response, and let $\mathcal{S}_{\text{batch}}$ be the set of n_{batch} cut states. For each problem in $\mathcal{S}_{\text{refill}}$, we do a single full rollout to the end, and add these to $\mathcal{B}$, which is simply a first-in-first-out queue with fixed capacity. We only use these rollouts to update the buffer, and they are not part of the actor and critic loss.

We use the trajectory prefixes in $\mathcal{S}_{\text{batch}}$ to update the actor and critic. We partition the problems into ready and unready sets according to whether they have satisfied the readiness criteria, and write $\mathcal{D}_{\text{ready}} \subseteq \mathcal{D}$ for the ready set. A prefix is ready when its problem is. By default, a problem is not ready. Assume we have access to the error of the critic ε on the problem from prior steps (we explain how this error is calculated in the following paragraph). A problem is deemed ready when three conditions hold: (1) the mean of ε over every prefix sampled in the last five steps falls below

Algorithm 1 Actor-Critic with Action Chunking (AC2)

Require: problems $\mathcal{D}$, actor π_θ and critic V_θ^π, group size g, chunk length b

$\mathcal{B} \leftarrow \emptyset$, $\mathcal{D}_{\text{ready}} \leftarrow \emptyset$
for iteration $k = 1, 2, \ldots$ **do**
  Sample $\mathcal{S}_{\text{refill}} \subseteq \mathcal{D}$; cut $\mathcal{S}_{\text{batch}}$ from rollouts in $\mathcal{B}$
  for each $x \in \mathcal{S}_{\text{refill}}$ **do**
    Generate one full response, judge it, insert into $\mathcal{B}$
  for each $s \in \mathcal{S}_{\text{batch}}$ **do**
    Sample $c_1, \ldots, c_g \sim \pi_\theta(\cdot \mid s)$, to termination if the problem $\notin \mathcal{D}_{\text{ready}}$, else for at most b tokens
    $v_i \leftarrow V_\theta^\pi(s \cdot c_i)$ if $s \cdot c_i$ is not complete, else r_i, for $i = 1, \ldots, g$
    $\hat{A}_i \leftarrow v_i - (\sum_j v_j)/g$
    $\varepsilon \leftarrow \left| V_\theta^\pi(s) - (\sum_j v_j)/g \right|$
  Add to $\mathcal{D}_{\text{ready}}$ each problem satisfying both readiness criteria based on $\varepsilon(s)$ (section 3.1)
  Update actor (Equation 5) using $\hat{A}_i$ and critic

a global threshold τ_{global}, (2) the error recorded for that problem from the prior step it was sampled falls below a tighter local threshold τ_{local}, and (3) at least one correct trajectory has been found by the policy on this problem.

For each prefix $s \in \mathcal{S}_{\text{batch}}$ we then sample a group of g continuations $c_i \sim \pi_\theta(\cdot \mid s)$ from the policy. If the problem is not ready, each continuation runs until the response terminates and receives a terminal reward r_i. In this case we use the terminal reword to calculate the critic error $\varepsilon = \left| V_\theta^\pi(s) - \text{mean}_i(r_i) \right|$ and store it for future readiness calculations. If the problem is ready, each continuation is a single chunk of at most b new tokens, after which generation stops. For a fraction α of ready problems (normally $\alpha = 1/4$) we rollout full trajectories, so that terminal rewards continue to be recorded for them and ε remains measurable after a problem becomes ready. This is useful for (1) measuring the accuracy of the critic during the run and (2) for updating the critic on ready problems (see section 3.3). We call this feature *auditing*, and it can be turned off by setting $\alpha = 0$ (see section 4.3 for this ablation).

Overall this gives us, for every $s \in \mathcal{S}_{\text{batch}}$, a group of g continuations $\{c_i\}_{i=1}^g$ and hence g new states $\{s \cdot c_i\}_{i=1}^g$. We write $\mathcal{G} = \left\{(s, \{c_i\}_{i=1}^g) : s \in \mathcal{S}_{\text{batch}}\right\}$ for the set of prefix–group pairs. We use $\mathcal{G}$ to compute actor and critic updates.

3.2 Advantage estimation

We use every pair $(s, \{c_i\}) \in \mathcal{G}$ to update the actor and critic. For each pair we wish to compute an advantage $\hat{A}_i$ for continuation c_i. By Equation 1, the advantage of a block is $V^\pi(s \cdot c_i) - V^\pi(s)$. We first consider the expected return $V^\pi(s \cdot c_i)$ for each new state $s \cdot c_i$. If $s \cdot c_i$ is a complete trajectory we know this exactly, is it the terminal reward r_i. If the trajectory is not complete, which is usually the case on critic ready problems that we sample a single action chunk for, we we use the critic, $V_\theta^\pi(s \cdot c_i)$. Write v_i for this endpoint value at $s \cdot c_i$, whether it is the terminal reward or the critic's estimate. The advantage also needs $V^\pi(s)$, the expected return from the prefix itself. Since $V^\pi(s) = \mathbb{E}_{c \sim \pi}\left[V^\pi(s \cdot c)\right]$ the group-mean value $\text{mean}_j(v_j)$ is an unbiased Monte-Carlo estimate of it. Together these give $\hat{A}_i = v_i - \frac{1}{g}\sum_{j=1}^g v_j$.

Note that on critic ready problems, this is the the $\lambda = 0$ form of the GAE advantage in Equation 4, with the baseline estimated from the group. On an unready problem, every v_j is a terminal reward, and thus the advantage is the $\lambda = 1$ form. Readiness can thus be interpretted as a per-problem switch to the λ parameter of the advantage estimate. With $\lambda = 0$ we *no longer need to observe the terminal reward*, and instead can update on a single action-chunk.

The critic V_θ^π share weights with the policy network. We queried the critic through a prompt rather than a separate head. The prompt contains the problem, the partial response, and, when the problem has previously been solved, a known-correct solution. The model answers with a value in $\{0, 0.1, \ldots, 1\}$, which we decode greedily and parse. The exact instruction is given in Appendix A.6.

3.3 ACTOR AND CRITIC UPDATE

The actor optimizes objective of equation 5 on the newly generated tokens, holding $\hat{A}_i$ fixed across the tokens of c_i. Replayed prefix tokens are conditioning context and receive no loss. We adapt ϵ_{high} based on the policy entropy following The Microsoft AI Team (2026).

The critic's target for a prefix is the group-mean value $\text{mean}_j(v_j)$, rounded to the nearest grid value. Since the critic answers with a value on a discrete grid, we fit it is as a next-token-prediction task. When the problem has a reference solution we train the same target under both prompts, with and without the reference, at equal weight. When there is no reference solution we train only the plain prompt. Prefix–target pairs are kept in a first-in-first-out buffer of their own and each critic step trains on a batch sampled from it, so a target is reused across several updates before it ages out.

We optimize the same parameters θ for both objectives, using different optimizer state and learning rate for each. For all our experiments, at each step we update the actor with two steps. We interleave the critic update between the two. We defer details of this algorithm to Section A.6.

4 EXPERIMENTS

We test AC2 by training Qwen3-4B-Thinking-2507 on FineProofs-RL (LM-Provers et al., 2026), a set of roughly 5,200 Olympiad proof problems from international competitions and AoPS. We evaluate on IMO-ProofBench (Luong et al., 2025), 60 Olympiad problems curated with grading rubrics and reference solutions. We use DeepSeek-V4-Flash (DeepSeek-AI et al., 2026) to judge proof correctness, scoring each generated proof from 0 to 7. The judge is prompted with the correct reference solution and problem-specific rubric as well as the candidate proof to grade for validation. We do not provide the reference solution or the problem-specific rubric to the judge during training.

Methods. We compare AC2 with GRPO and Prefix GRPO, a GRPO variant that starts its rollouts from replayed prefixes. AC2 uses group size $g = 16$, chunk length $b = 10{,}000$, and auditing with $\alpha = 1/4$, and empty initial replay buffers. Prefix GRPO uses the same actor trajectory buffer system as AC2, but rolls out full-length continuations of prefixes and uses terminal rewards for advantage estimation identical to GRPO. We select the GRPO learning rate by peak validation mean score (Appendix A.1). See Table 5 and hyperparameters and Appendix A.6 for additional details.

Compute accounting. We track policy performance against a measure of compute used. The standard measure would be number of steps. This is imperfect in our setting as AC2, Prefix GRPO, and GRPO use different amounts of compute, and accordingly wall-clock time for each step. Specifically, GRPO uses the most compute as it requires rolling out all trajectories to completion. A strong candidate to use is GPU hours (as used by Khatri et al. (2026)), however we conduct our experiments using different numbers of GPUs, and due to differing communication overhead GPU hours are not directly comparable between runs. Instead we find Decoding FLOPs (the FLOPs used in policy rollouts) correlate strongly with GPU hours in our main run (Appendix A.5) and so use this hardware-agnostic measure instead. We also report performance against steps for full clarity.

4.1 MAIN RESULTS

AC2 is more compute efficient and data efficient than GRPO. We present results in Figure 1. AC2 reaches higher validation score with fewer steps and less decoding compute than GRPO (Figure 1). AC2 first exceeds GRPO's peak mean score of 18.50% at 0.79e20 FLOPs compared to 1.99e20 for GRPO. This is a $2.5\times$ compute efficiency gain over GRPO. Here both algorithms are also acting under a data constrained setting as there is a fixed number of training problems that we epoch. Under this constraint, AC2 reaches a higher peak score of 20.57% over 18.5% for GRPO.

The replay-buffer state distribution alone does not explain AC2's gains. We test whether AC2's gains come from the state distribution induced by replay using Prefix GRPO. Prefix GRPO falls well behind AC2 when measured by Decoding FLOPs (Figure 1, right).

The critic becomes trustworthy early and stays so. The first three panels of Figure 2 track the two quantities that govern how much AC2 relies on its critic. The global readiness threshold is first crossed at the end of step 7, after which the fraction of sampled problems that are ready rises steadily, reaching about 70% by step 200. The critic's error ε decrease steadily over training.

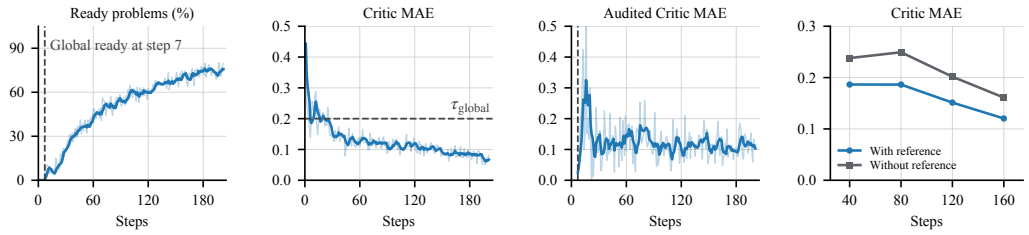

Figure 2: **Critic readiness and accuracy for AC2.** The first three panels show critic metrics over the course of the main AC2 run, shown as the blue line in Figure 1. First: fraction of sampled problems that are ready. Second: critic error over all sampled prefixes, $\left|V_\theta^\pi(s) - \text{mean}_j(v_j)\right|$, where v_j is the terminal reward or the critic's value of the continuation. Third: critic error on audited groups of critic ready problems, $\left|V_\theta^\pi(s) - \text{mean}_j(r_j)\right|$, where every continuation runs to completion, so the target is a mean of terminal rewards only. In the first three panels, faint lines show per-step values, averaged over that step's prefixes in the second and third panels. Dark lines average these over the trailing five steps. Fourth: critic MAE with and without the reference solution in its prompt.

4.2 AC2 VARIANTS

We present three variants of the AC2 algorithm that also perform well: (1) training with a highly stale replay buffer, (2) training without auditing, and (3) setting group size on ready problems to 1. Results are shown in Figure 3, left.

AC2 with stale replay buffer. Actor–critic methods (Haarnoja et al., 2018; Fujimoto et al., 2018), such as Soft Actor-Critic (SAC), can learn from rollouts generated by earlier policies. We test AC2s tolerance to a more stale initial state distribution by collecting 1920 fresh rollouts every 10 steps and setting this to be the actor replay buffer for those steps (instead of 192 rollouts per step and a buffer with size 256). The mean sore reaches 17.90% and appears to plateau there, but reaches this value far faster than GRPO. We conclude that AC2 can learn from a more heavily stale state distribution.

AC2 without auditing. We try removing auditing from critic ready problems (Figure 3, teal line). The run shows slightly larger instability than AC2 run, but it still reaches a peak mean of 18.66% at step 160, outperforming the GRPO baseline. We still recommend using auditing as it reaches a higher peak reward and allows for tracking the health of the critic during training more closely.

AC2 with group size 1. We next consider removing grouping as well as auditing, giving AC2 w/o Group & Audit. For each sampled ready problem, this variant samples 16 separate prefixes and generates one action chunk c_i per prefix. Each continuation uses the advantage estimate $\hat{A}_i = v_i - V_\theta^\pi(s)$, with $v_i = V_\theta^\pi(s \cdot c_i)$ or $v_i = r_i$ if the action chunk ends the trajectory. The critic is trained on the same single endpoint, fitting $V_\theta^\pi(s)$ to v_i as per the Bellman equation. We retain full-length groups on unready problems to train the critic using average terminal rewards. This run performs similarly to AC2 (Figure 3, purple line).

4.3 ABLATIONS THAT HURT PERFORMANCE

We now ablate the other components of AC2 to show their necessity. We find that the use of the local readiness criterion and 10k long action chunks (as opposed to smaller ones) are essential. Intuitively, local readiness better ensures the critic is accurate when we use it, and larger action chunks are easier for a the critic to judge, and thus make it more accurate. Finally, we consider only storing correct trajectories in the actor replay buffer like Setlur et al. (2026), but find this hurts performance.

Smaller chunk size hurts performance. We test AC2 with chunk size of 2,000 as opposed to 10,000 used in the main run. The two runs track each other through step 50, where AC2 w/ 2k chunks reaches 16.03%, compared with 16.41% for AC2's 10,000-token chunks. Beyond that step the shorter chunks stop improving. AC2 w/ 2k chunks peaks at 16.62% at step 100 and falls to 13.57% at step 120, where AC2 reaches 19.40% (Figure 3, green line).

AC2 w/ correct-only buffer improves early scores but worsens the later trajectory. Motivated by Setlur et al. (2026), we keep only trajectories receiving at least six of seven training-judge points

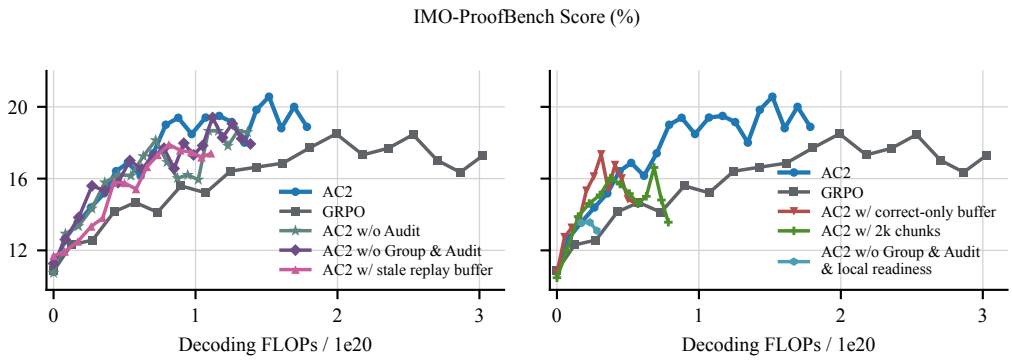

Figure 3: **Component ablations evaluated by mean score versus Decoding FLOPs.** Left: variants that perform comparably to AC2. AC2 w/o Audit removes full-length audits with little effect on performance. AC2 w/o Group & Audit uses one short continuation per ready prefix and the starting-prefix value as its baseline, achieving comparable scores. AC2 w/ stale replay buffer refreshes its replay buffer only every ten steps and continues to improve. Right: variants that fall behind. With AC2 w/o Group & Audit & local readiness, learning stalls and scores decline; this run branches off AC2 at the checkpoint of step 20. AC2 w/ correct-only buffer retains trajectories receiving at least six of seven judge points in the replay buffer, following Setlur et al. (2026), and worsens later in training. AC2 w/ 2k chunks uses 2,000-token chunks and falls behind after step 50. The corresponding training-step plots are in Appendix Figure 6.

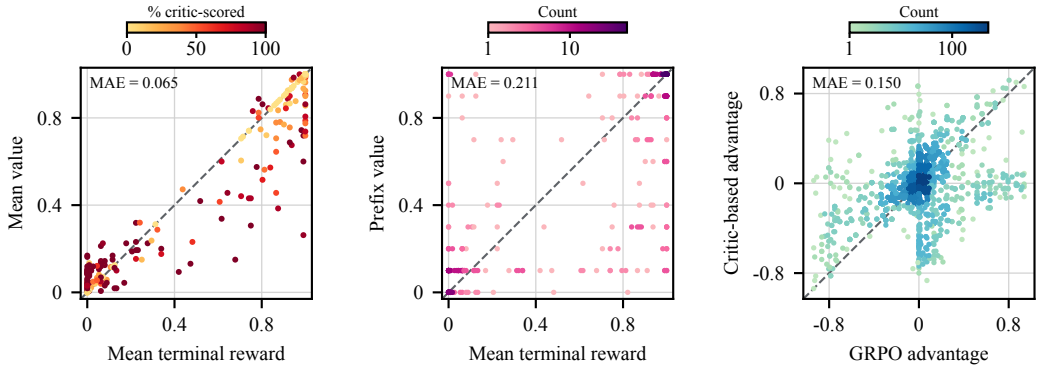

Figure 4: **Group-mean values are more accurate than prefix predictions, and critic-based advantages correlate with GRPO advantages.** At step 80, the left panel compares $\text{mean}_i(v_i)$ with $\text{mean}_i(r_i)$ for 256 prefixes; colour indicates the percentage of continuations scored by the critic. The middle panel compares the prefix value prediction $V_\theta^\pi(s)$ with $\text{mean}_i(r_i)$ for the same prefixes. The right panel compares $\hat{A}_i$ with $r_i - \text{mean}_j(r_j)$ for 2,640 responses in 165 groups containing at least one critic prediction. Details are deffered to Appendix A.4.

in the replay buffer. This run reaches 17.35% at step 60, compared with 16.88% for AC2. However, this early advantage does not persist: at step 110, its score falls to 14.67% (Figure 3, red line).

Without local readiness. We now consider ablating the use of a local readiness threshold. Specifically, as soon as the critic MAE drops below the global τ_{global} threshold, we consider all problems critic ready. For this ablation we run without group and audit. Specifically we branch off the AC2 run at step 20 (the blue line in Figure 3). With local readiness disabled, mean score falls from 13.56% at step 30 to 13.10% at step 40, while the run without group and audit but with local readiness reaches 15.33% at step 40 (Figure 3, cyan line versus purple line). At that step, only 18.23% of sampled problems are ready with local readiness, compared with 100% without it (Appendix Figure 9).

4.4 CRITIC DIAGNOSTICS

Group-mean values are more accurate than the prefix value. At step 80, we sample 256 ready training problems, cut one prefix s from each problem's latest stored trajectory, and generate $g = 16$ full continuations. We reconstruct the endpoint value v_i using the critic prediction after 10,000 new tokens when available and the terminal reward r_i otherwise (Appendix A.4). Against the mean terminal reward $\text{mean}_i(r_i)$, the group-mean value $\text{mean}_i(v_i)$ has mean absolute error (MAE) 0.065, compared with 0.211 for the prefix value $V_\theta^\pi(s)$ (Figure 4, left and middle). This justifies using the group-mean value as supervision for the prefix value.

Critic-based advantages are positively correlated with GRPO advantages. Although the prefix value has a larger MAE, the critic-based advantages $\hat{A}_i = v_i - \text{mean}_j(v_j)$ have MAE 0.150 relative to the GRPO advantages $r_i - \text{mean}_j(r_j)$ across 2,640 responses in 165 groups containing at least one critic prediction (Figure 4, right). Their Pearson correlation is 0.388. As AC2 outperforms Prefix GRPO (Section 4.1), GRPO advantages should not be treated as ground truth here: Prefix GRPO yields worse policy performance in this comparison.

A reference solution in the prompt makes the critic more accurate. We take the main run's checkpoints (the blue line in Figure 1) at steps 40, 80, 120 and 160. For each, we collect the prefixes probed over the next 20 steps whose problem had a reference solution and whose target is a mean of terminal rewards, about 1,500 per checkpoint. The critic scores each prefix twice, with and without the reference in its prompt. With the reference, MAE is lower at every checkpoint, by 0.04 to 0.06 (Figure 2, fourth panel), falling from 0.238 to 0.187 at step 40 and from 0.161 to 0.120 at step 160.

5 RELATED WORK

Reinforcement learning with verifiable rewards (RLVR) typically uses methods such as GRPO (Shao et al., 2024), which generate complete responses and compute advantages from terminal rewards. Recent work reintroduces learned critics, often primarily as baselines for these rewards, including EVPO (Pan et al., 2026), GenAC (Shan et al., 2026), and the concurrent work Le Critique (Venkatraman et al., 2026). Our work highlights a further use of the critic: once it is sufficiently accurate on a problem, AC2 can update the policy using only critic scores of unfinished continuations, without rolling out to a terminal reward. Using the critic so aggressively requires it to be accurate. We achieve this using three components. First, *local readiness* determines whether the critic is accurate enough on each problem to replace terminal rollouts with short continuations. This is related to previous works that used global criteria to evaluate critic accuracy (Pan et al., 2026; Venkatraman et al., 2026). Second, providing *privileged information*, a correct solution only to the critic helps value prediction, as also explored in Venkatraman et al. (2026); Qi et al. (2026). Third, *action chunking* provides an intermediate unit of credit assignment between individual tokens and complete responses, similar to previous attempts in Li et al. (2025); Guo et al. (2025); Kazemnejad et al. (2025). Appendix B provides an extended discussion.

6 DISCUSSION

We forward the idea that critics in LLM RL can be used more aggressively than prior works. One example of this is abandoning the requirement of collecting a terminal reward, instead using a GAE advantage with $\lambda = 0$. Removing this requirement opens up the design space of LLM RL algorithms, with AC2 being an example of such a new algorithm. AC2 also leverages the special property that you can reset the states during LLM RL by sampling action chunks on top of prefixes from a replay buffer. One can imagine more exotic (to the modern reader) combinations of a critic and state resetting. For example, you could conduct beam search over action chunks using an AC2 trained model, or even alpha-go style MCTS tree search (Silver et al., 2016).

One of the key components that makes AC2 work, local readiness, is also its largest limitation. In particular, local readiness requires epoching the training data as we require prior group rollouts to completion to get a "ground truth" estimate of the value. In the non data constrained setting, this epoching is not realistic. We leave adapting AC2, and more generally the local readiness criteria, to the single epoch regime as future work.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods and interpreting results. We have not used generative AI tools for all other tasks listed on the required disclosure. Additionally, we used generative AI tools for creating figures, initial drafting of the manuscript, summarizing pre-existing literature . We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness at the thoroughness of human authored code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 3 describes our algorithm, and Section 4 specifies the datasets and evaluation protocol. Appendix A.6 and Table 5 provide implementation details and hyperparameters; Appendix A.5 documents our compute accounting. We provide an anonymized supplementary archive containing the training code, experiment configurations, data-preparation scripts, and reproduction instructions. The archive also includes recorded measurements and plotting scripts for the performance and ablation results.

REFERENCES

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.

Yunho Choi, Jongwon Lim, Woojin Ahn, Minjae Oh, Jeonghoon Shim, and Yohan Jo. Your language model is its own critic: Reinforcement learning with value estimation from actor's internal states. *arXiv preprint arXiv:2605.07579*, 2026.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu,

Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. Pmlr, 2018.

Yiran Guo, Lijie Xu, Jie Liu, Dan Ye, and Shuang Qiu. Segment policy optimization: Effective segment-level credit assignment in RL for large language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3815. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a6536243037d1e32c20de85137d478da-Abstract-Conference.html.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Refining credit assignment in RL training of LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 29557–29590. PMLR, 2025. URL https://proceedings.mlr.press/v267/kazemnejad25a.html.

Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Venkata Sai Surya Subramanyam Duvvuri, Manzil Zaheer, Inderjit Dhillon, David Brandfonbrener, and Rishabh Agarwal. The art of scaling reinforcement learning compute for llms. In *International Conference on Learning Representations*, volume 2026, pp. 72438–72467, 2026.

Qiyang Li, Zhiyuan Zhou, and Sergey Levine. Reinforcement learning with action chunking. *arXiv preprint arXiv:2507.07969*, 2025. URL https://arxiv.org/abs/2507.07969.

LM-Provers, Yuxiao Qu, Amrith Setlur, Jasper Dekoninck, Edward Beeching, Jia Li, Ian Wu, Lewis Tunstall, and Aviral Kumar. QED-Nano: Teaching a tiny model to prove hard theorems. *arXiv preprint arXiv:2604.04898*, 2026. URL https://arxiv.org/abs/2604.04898.

Thang Luong, Dawsen Hwang, Hoang H Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, Alex Zhai, Huiyi Hu, Henryk Michalewski, Jimin Kim, Jeonghyun Ahn, Junhwi Bae, Xingyou Song, Trieu Hoang Trinh, Quoc V Le, and Junehyuk Jung. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35418–35442. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1794. URL https://aclanthology.org/2025.emnlp-main.1794/.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PmLR, 2016.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser

Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Chengjun Pan, Shichun Liu, Jiahang Lin, Dingwei Zhu, Jiazheng Zhang, Shihan Dou, Songyang Gao, Zhenhua Han, Binghai Wang, Rui Zheng, et al. Evpo: Explained variance policy optimization for adaptive critic utilization in llm post-training. *arXiv preprint arXiv:2604.19485*, 2026.

Penghui Qi, Xiangxin Zhou, and Wee Sun Lee. Best practice critic optimization. *arXiv preprint arXiv:2608.23566*, 2026.

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

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. URL https://arxiv.org/abs/1506.02438.

Amrith Setlur, Zijian Wang, Andrew Cohen, Paria Rashidinejad, and Sang Michael Xie. Reuse your FLOPs: Scaling RL on hard problems by conditioning on very off-policy prefixes. *arXiv preprint arXiv:2601.18795*, 2026. URL https://arxiv.org/abs/2601.18795.

Zikang Shan, Han Zhong, Liwei Wang, and Li Zhao. Bringing value models back: Generative critics for value modeling in llm reinforcement learning. *arXiv preprint arXiv:2604.10701*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1999.

The Microsoft AI Team. MAI-Thinking-1: Building a hill-climbing machine. Technical report, Microsoft AI, 2026. URL https://microsoft.ai/pdf/mai-thinking-1.pdf.

Siddarth Venkatraman, Matthieu Dinot, and Laurence Aitchison. Le critique: Privileged value functions for llm reinforcement learning. *arXiv preprint arXiv:2608.16739*, 2026.

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.

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. *arXiv preprint arXiv:2503.14476*, 2025.

Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, et al. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*, 2025.

Yi-Kai Zhang, Yueqing Sun, Hongyan Hao, Qi Gu, Xunliang Cai, De-Chuan Zhan, and Han-Jia Ye. $V_{0.5}$: Generalist value model as a prior for sparse RL rollouts. *arXiv preprint arXiv:2603.10848*, 2026.

Table 1: GRPO learning-rate selection by peak observed mean score. Bold marks the selected learning rate. Training durations differ.

Learning rate	Last step	Peak mean score (%)
1e-6	70	15.40
2e-6	180	**18.50**
4e-6	100	16.95

A Additional experiments and measurement details

A.1 GRPO tuning

We select the GRPO learning rate by the highest observed validation mean score among three candidates. Table 1 and Figure 5 show the sweep. The 2×10^{-6} run has the largest observed peak, 18.50%, and is used in the main comparison. The candidates have unequal durations, so the table records the last evaluated step. Selection uses the reported validation trajectories rather than an independent tuning split. All three score trajectories use the original evaluation metrics.

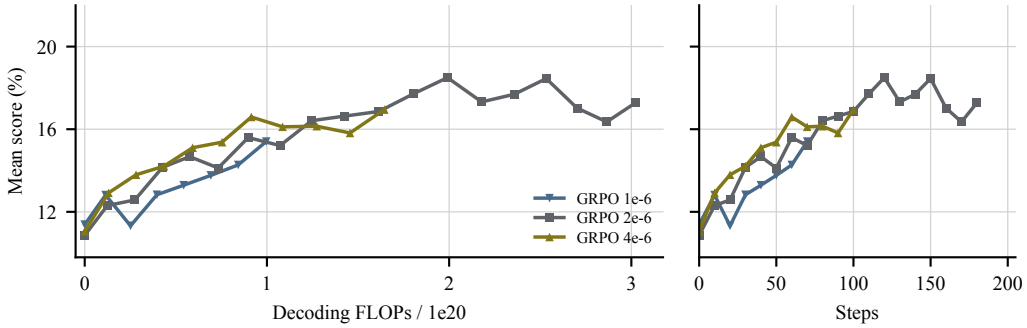

Figure 5: **The observed learning-rate sweep favors 2×10^{-6}.** Panels show mean score against (a) Decoding FLOPs and (b) Steps. Each curve covers all evaluated checkpoints.

A.2 Additional ablations

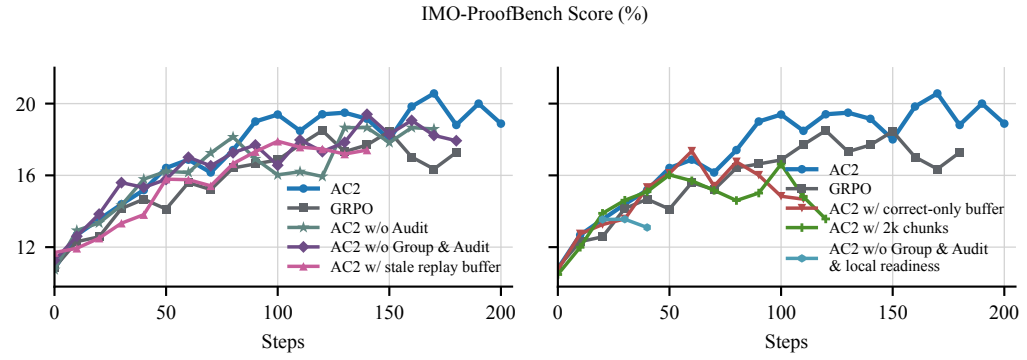

Figure 6: **Component ablations evaluated by mean score versus training steps.** The panels show the comparisons from Figure 3 with training steps on the horizontal axis. Left: variants that perform comparably to AC2. Right: variants that fall behind. Colors and markers match the main figure.

The audit comparison depends on training history. The no-audit run from initialization reaches 18.66% at step 160, compared with 19.84% for the main run (Figure 7). This differs from the shared-checkpoint comparison, where the no-audit branch slightly exceeds the main run at step 120. The two experiments test different training histories and do not establish a consistent ordering across initialization choices.

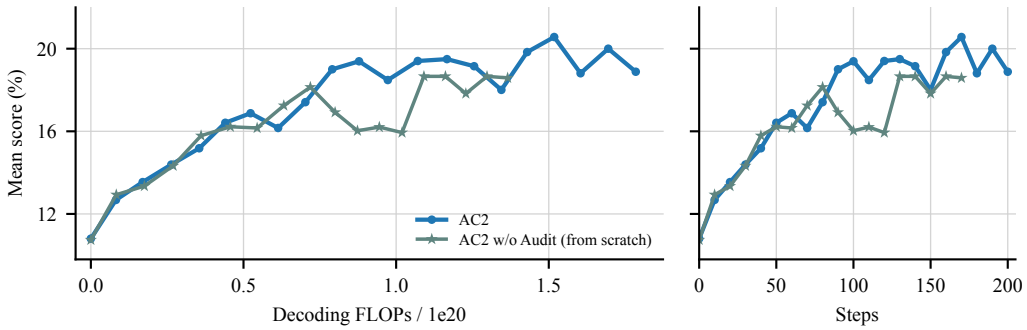

Figure 7: **The audit comparison depends on training history.** Both runs begin from the base model. Axes are (a) Decoding FLOPs and (b) Steps.

Starting-prefix baseline. Figure 3 compares AC2 with AC2 w/o Group & Audit. Full groups retain 16 continuations per prefix. For each sampled problem assigned to short generation, AC2 w/o Group & Audit samples 16 separate prefixes and generates one continuation from each ($g = 1$). This preserves the total number of continuations per sampled problem. The short-prefix value targets use individual endpoint values instead of group-mean values; full groups still supply grouped targets.

Local and global readiness. The local gate retains the problem-specific eligibility test in Section 3.1; the global gate uses the pooled error criterion to route all problems as ready. Both readiness variants use one continuation per prefix for short updates and disable auditing. The local run starts from the base model with empty buffers. The global run branches off AC2 at the checkpoint of step 20, following audited, grouped training. It then removes grouping for short updates, auditing, and local readiness. Its global gate is closed at that checkpoint, but the parent's gate had opened earlier, so the inherited state is not equivalent to a fresh initialization. Figure 3 includes the parent's first 20 steps in the global trajectory. Figure 9 shows the corresponding ready-routing fractions. The local run reaches 17.26% mean score at step 80; global-run evaluations end at step 40.

Chunk size. We compare AC2 w/ 2k chunks with AC2, using grouping and auditing in both runs (Figure 8). Both runs start from the base model. AC2 w/ 2k chunks also reduces the spacing between possible prefix cuts to 2,000 tokens. At step 50, AC2 w/ 2k chunks and AC2 reach mean scores of 16.03% and 16.41%, respectively. Their cumulative decoding costs are 3.77e19 (estimated) and 4.40e19 FLOPs. The two runs separate afterwards. AC2 w/ 2k chunks peaks at 16.62% at step 100 and ends at 13.57% at step 120, its last evaluation, having spent 7.85e19 estimated FLOPs; AC2 reaches 19.40% at the same step at a cost of 1.07e20 FLOPs, and continues to 20.57% at step 170. Because this configuration changes the continuation length and the prefix-cut spacing together, the comparison does not isolate either change.

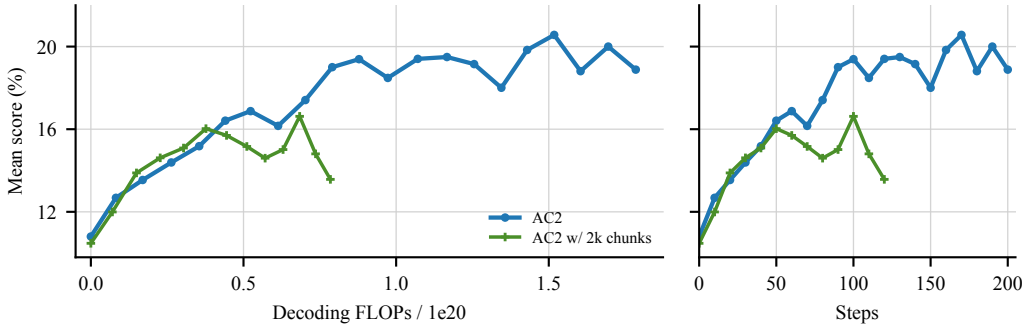

Figure 8: **Chunk-size ablation with grouping and auditing enabled.** AC2 uses 10,000-token chunks; AC2 w/ 2k chunks uses 2,000-token chunks and prefix-cut spacing. Both start from the base model. AC2 w/ 2k chunks has evaluations through step 120. Axes are (a) Decoding FLOPs and (b) Steps.

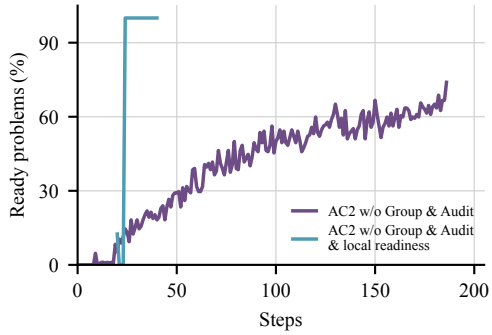

Figure 9: **Removing local readiness makes all sampled problems eligible after global readiness.** AC2 w/o Group & Audit starts from the base model; the variant that also removes local readiness branches off AC2 at the checkpoint of step 20. The plot shows their sampled-problem ready fractions.

The longer response limit produces little score separation over the recorded continuation. Increasing the response limit from 50,000 to 75,000 tokens at step 130 produces little separation over the recorded continuation (Figure 10). At step 160, the branch reaches 20.04%, compared with 19.84% for the main run. This short comparison does not establish a systematic benefit from the longer limit; it also changes the permitted evaluation response length.

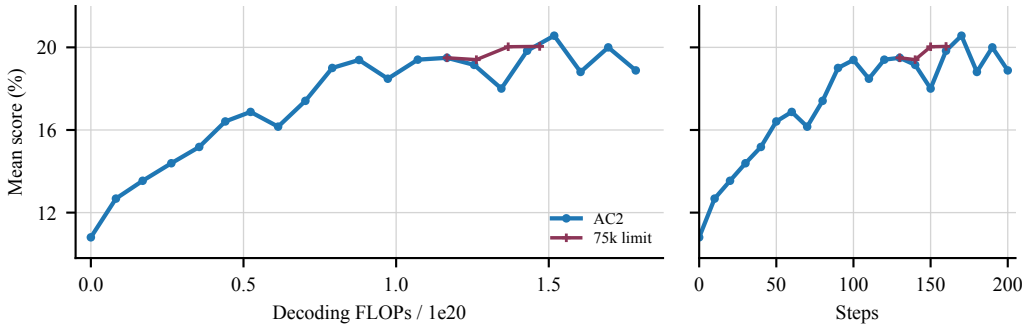

Figure 10: **The retained longer-limit branch shows little score separation over its observed horizon.** The 75,000-token branch begins at step 130. Axes are (a) Decoding FLOPs and (b) Steps.

A.3 Additional evaluation metrics

To compute the best-of-16 normalized score, the logging procedure draws, for each problem, 1,000 bootstrap samples of size 16 with replacement from its 16 evaluated scores, computes the maximum in each sample, and averages these maxima. It then averages over problems and reports a percentage. This is a bootstrap estimate of a maximum normalized grade, rather than a binary success rate. The main run's score is 32.36% at step 200; the selected GRPO run ends at 35.01% at step 180. These unequal-duration endpoints complement the trajectories rather than define an exact-budget comparison.

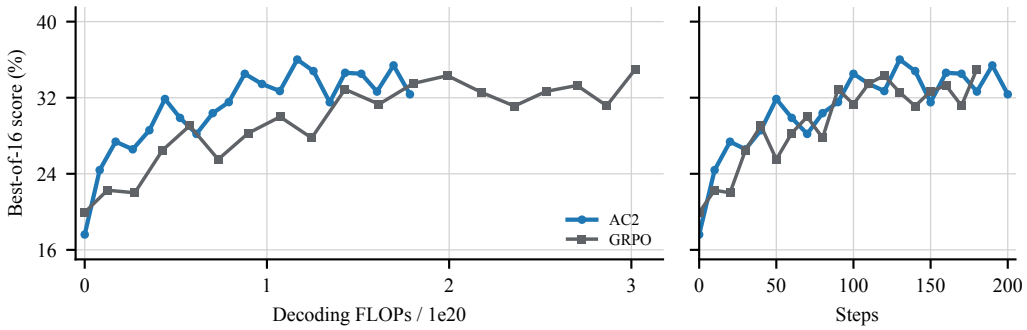

Figure 11: **Best-of-16 normalized scores provide a complementary view of generated proof quality.** Values are bootstrap means of maximum normalized grades. Axes are (a) Decoding FLOPs and (b) Steps.

Validation response length measures how long the evaluated model's outputs are, separately from the training requests used for compute accounting (Figure 12). The no-audit branch averages 29,213 response tokens at step 120. This diagnostic describes the generation behavior of the trained policy; it cannot replace the prompt, replay-prefix, and generated lengths of training rollouts in the decoding-cost calculation.

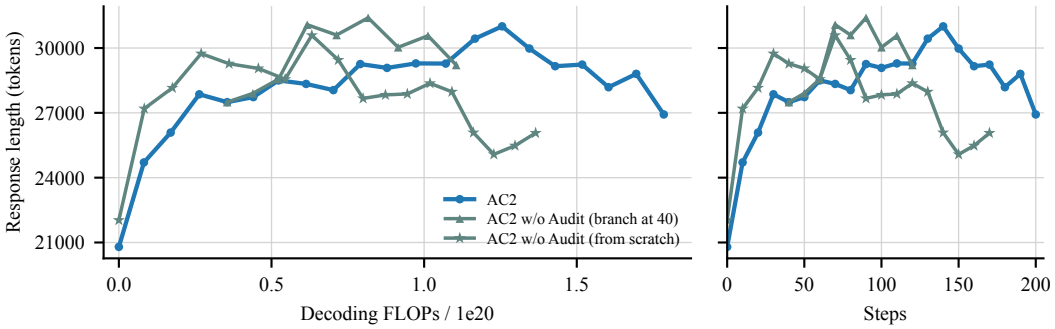

Figure 12: **Validation response length characterizes output behavior separately from training cost.** The no-audit branch starts at step 40. Axes are (a) cumulative training Decoding FLOPs and (b) Steps; the vertical axis measures evaluation response tokens.

A.4 Additional value function probes

We extend the probes in Section 4.4 to additional checkpoints. At each checkpoint, the diagnostic set selects 256 distinct problems marked ready and their latest available trajectories. Each source attempt is cut at a random fraction between 5% and 95%, with the prefix clamped to leave at least 12,000 response tokens. A frozen policy generates 16 full-budget continuations per prefix; the value function is queried at a further 10,000 tokens for continuations exceeding that length. For each prefix s, we reuse Method's notation: r_i is continuation i's terminal reward and c_i contains its first 10,000 new tokens if it reaches the cut, or its full continuation otherwise. The endpoint value v_i used to construct the advantage is reconstructed as

$$v_i = \begin{cases} V_\theta^\pi(s \cdot c_i), & \text{if a cut-state prediction is available,} \\ r_i, & \text{otherwise.} \end{cases} \tag{6}$$

We compute the group means as $\text{mean}_i(v_i)$ and $\text{mean}_i(r_i)$, and the advantages as $\hat{A}_i = v_i - \text{mean}_j(v_j)$ and $r_i - \text{mean}_j(r_j)$. Let f denote the fraction of a group's continuations with an available cut-state prediction. This hybrid reconstruction omits training's exemption for proofs completed before the cut; training judges such proofs instead. The resulting discrepancies are diagnostic rather than exact training errors. Diagnostic rollouts are excluded from training Decoding FLOPs. Figures 17–19 report matched-group and centered-signal comparisons at steps 40, 57, and 160. Figures 20 and 21 show individual cut-state predictions against their matched terminal rewards at all

four checkpoints. Groups in the matched-group plots contain at least two valid matched continuations. Centered-signal plots require all 16 responses and at least one reconstructed value substitution. Each checkpoint supplies its own probe population, so differences across these plots do not establish improvement on a fixed distribution.

The retained prefix-site records cover steps 57, 80, and 160. Each observation compares one $V_\theta^\pi(s)$ prediction with the mean normalized terminal reward of its 16 full-budget continuations. There are 248 valid paired records at step 57 and 256 at each later checkpoint. Step-40 cached summaries do not contain a prefix-value column. Unlike matched cut-state probes, this comparison does not condition on continuations reaching the cut. Prefix-value MAEs are 0.153, 0.211, and 0.173 at steps 57, 80, and 160, respectively. At step 80, the signed bias (mean prediction minus mean terminal reward across prefixes) is -0.040. These summaries use the retained CSV values, whose group means are rounded to four decimal places.

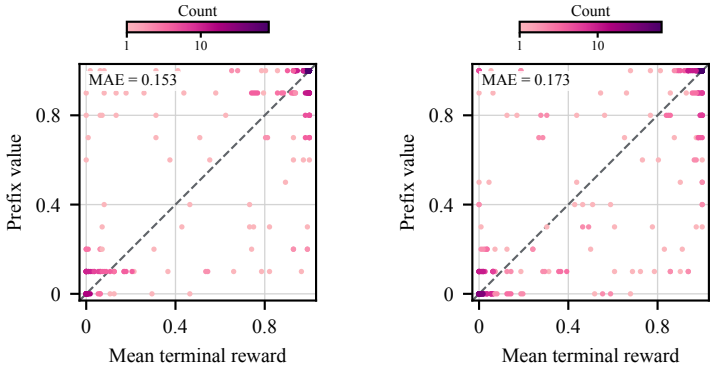

Figure 13: **Prefix-site value is compared with full-continuation group rewards.** Panels show steps (a) 57 ($n = 248$ prefixes) and (b) 160 ($n = 256$). Each x-value averages all 16 terminal rewards for a shared prefix; its y-value is the single value prediction at that prefix. Step 80 appears in Figure 4, middle. Dashed lines denote equality; color shows logarithmic counts.

Cut-state predictions and terminal rewards. Averaging cut-state predictions and their matched rewards within groups with at least two such continuations gives 157 groups, with MAE 0.169 and signed bias -0.094 at step 80 (Figure 14). This comparison uses different states and response sets from the prefix-value comparison in Figure effig:critic-diagnostics, middle. Individual cut-state MAE is 0.206 across 1,848 matched continuations (Figure 21(a)). Matched-group MAE is 0.312, 0.121, 0.169, and 0.159 at steps 40, 57, 80, and 160, respectively, so agreement does not improve monotonically.

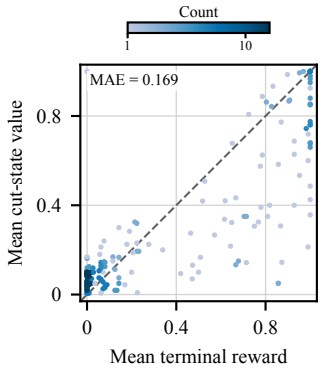

Figure 14: **Mean cut-state predictions differ from matched terminal rewards.** At step 80, the plot compares mean cut-state value with matched mean terminal reward for 157 groups containing at least two cut-reaching responses. The dashed line denotes equality; color shows counts on a logarithmic scale.

Group-mean error varies with the substitution fraction. Table 2 applies Equation 6 to complete groups at all four checkpoints. Groups with $f = 0$ agree by construction. At step 80, 91 groups use only terminal rewards and agree by construction. The 98 partially substituted and 67 fully substituted groups have MAEs of 0.096 and 0.106. Fully substituted group MAE ranges from 0.106 to 0.218 across checkpoints. These are different sampled populations, not a fixed-set calibration trajectory.

Table 2: Reconstructed group-signal errors by substitution fraction f. Each observation is a complete group of 16 continuations. Bias is reconstructed signal minus terminal reward, averaged over groups. Zero-substitution groups agree by construction.

Step	Substitution fraction	Groups	MAE	Signed bias
40	All	256	0.129	+0.112
40	$f = 0$	94	0.000	+0.000
40	$0 < f < 1$	79	0.188	+0.169
40	$f = 1$	83	0.218	+0.183
57	All	248	0.050	+0.008
57	$f = 0$	93	0.000	+0.000
57	$0 < f < 1$	103	0.066	+0.006
57	$f = 1$	52	0.107	+0.026
80	All	256	0.065	-0.031
80	$f = 0$	91	0.000	+0.000
80	$0 < f < 1$	98	0.096	-0.070
80	$f = 1$	67	0.106	-0.018
160	All	256	0.064	-0.041
160	$f = 0$	89	0.000	+0.000
160	$0 < f < 1$	99	0.079	-0.059
160	$f = 1$	68	0.126	-0.066

Advantage correlations. We compute correlations from the same within-group-centered pairs shown in Figure 4, right, pooled across groups with at least one critic prediction. At step 80, Pearson correlation is 0.388 and Spearman rank correlation is 0.362 across 2,640 responses. The two estimators share terminal rewards on continuations without a critic prediction. Restricting the comparison to the 1,848 critic-scored responses reduces Pearson correlation to 0.208; the 1,072 responses in fully critic-scored groups have correlation 0.157. We retain the original group means when selecting these subsets. These are descriptive correlations, without treating responses from the same group as independent observations.

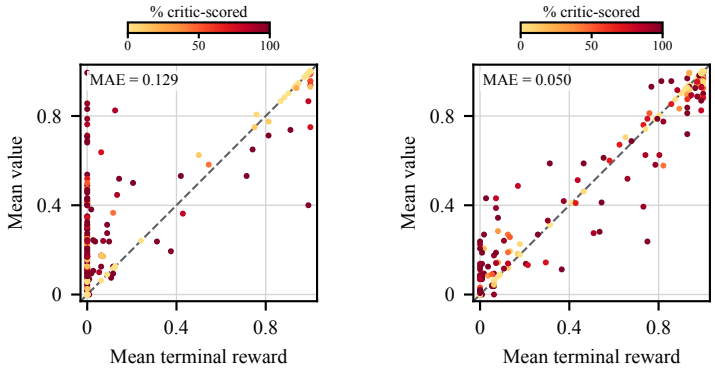

Figure 15: **Group-signal agreement depends on the fraction of value substitutions.** Panels show (a) step 40 and (b) step 57. Each point averages a complete group of 16 continuations; color indicates the percentage $100f$ replaced by available cut-state values. Dashed lines denote equality.

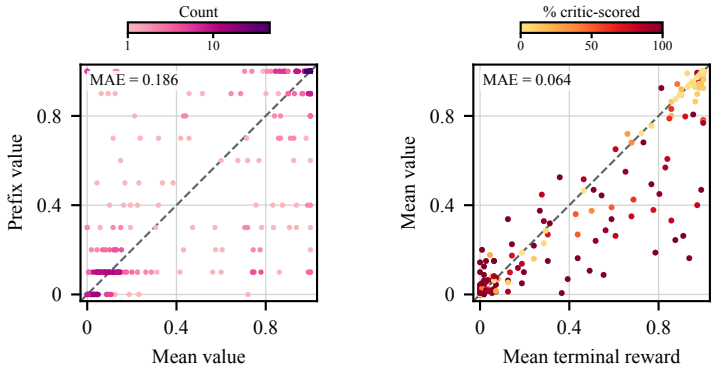

Figure 16: **Prefix predictions and reconstructed group signals answer different diagnostic questions.** (a) At step 80, prefix-site value versus the mean reconstructed hybrid signal for 256 prefixes, with logarithmic count coloring. (b) At step 160, reconstructed hybrid versus terminal group means for 256 prefixes, colored by $100f$. The reconstruction uses length-only substitution and omits production's proof-extraction exemption; it is neither an independent terminal target nor the exact rounded value-training target. Dashed lines denote equality. The step-80 centered comparison is in Figure 4, right.

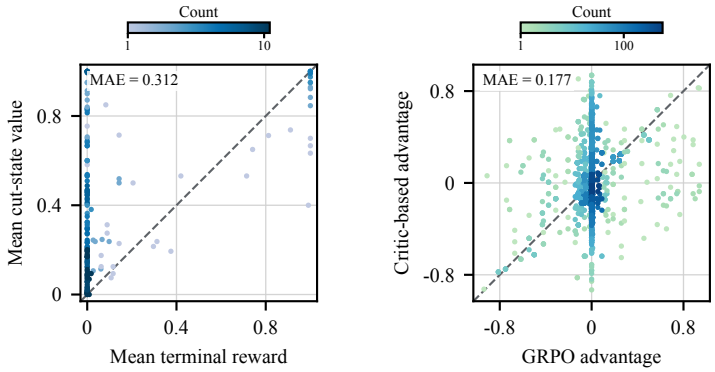

Figure 17: **Step-40 probes characterize the early value signals.** (a) Matched group means. (b) Centered signals in complete groups with value substitution. Dashed lines denote equality; color shows counts on a logarithmic scale.

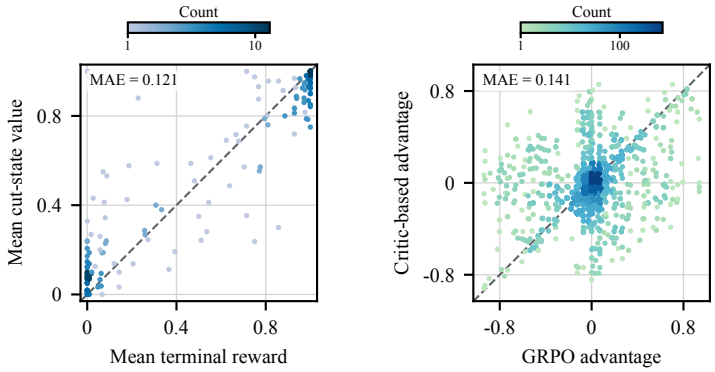

Figure 18: **Step-57 probes provide a second view of the value signals.** (a) Matched group means. (b) Centered signals. Dashed lines denote equality; color shows counts on a logarithmic scale.

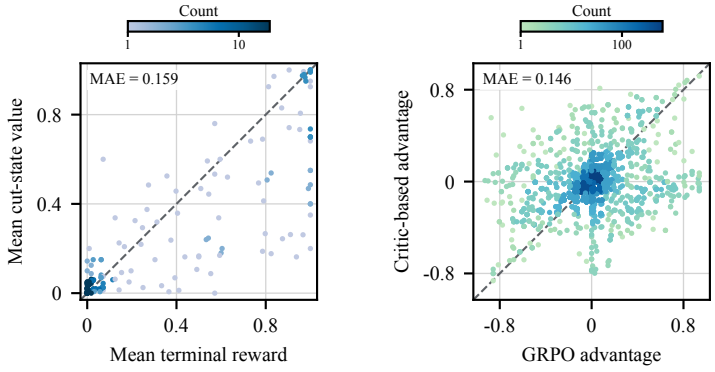

Figure 19: **Step-160 probes characterize the later value signals.** (a) Matched group means. (b) Centered signals. Dashed lines denote equality; color shows counts on a logarithmic scale.

Table 3: Matched-probe sample counts and absolute errors. Each checkpoint uses its own sampled prefixes. Group counts require at least two matched continuations; centered and individual counts are response counts.

Step	Group n	MAE	Centered n	MAE	Individual n
40	153	0.312	2592	0.177	1919
57	144	0.121	2480	0.141	1685
80	157	0.169	2640	0.150	1848
160	153	0.159	2672	0.146	1892

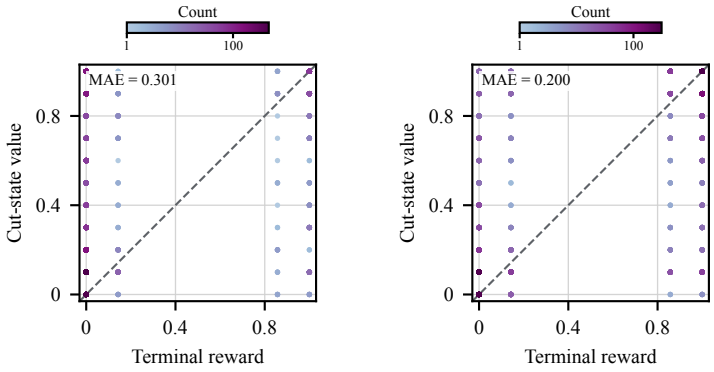

Figure 20: **Individual cut-state value estimates are compared with realized terminal rewards.** Panels show steps (a) 40 and (b) 57. Each observation is a matched continuation; color shows counts on a logarithmic scale.

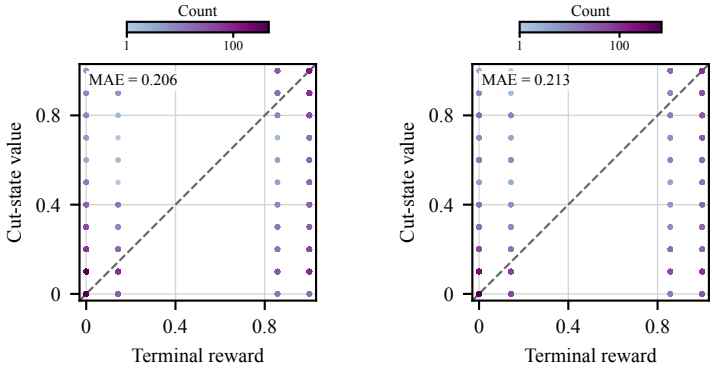

Figure 21: **Individual-value diagnostics remain distinct from the group-mean comparison.** Panels show steps (a) 80 and (b) 160. Dashed lines denote equality; color shows counts on a logarithmic scale.

A.5 Decoding FLOPs accounting and GPU hours

We compute decoding costs from joint prefix and generated-length statistics when available, and use per-step training-length means otherwise. AC2 uses joint length statistics throughout. All three GRPO runs, the local-readiness singleton run from initialization, and AC2 w/ 2k chunks use mean-length estimates throughout. Prefix GRPO, the no-audit run from initialization, AC2 w/ correct-only buffer, and AC2 w/ stale replay buffer combine means with retained joint length records. Table 4 lists the source and coverage of every run. These distinctions apply to all FLOPs plots and tables in the paper.

For a policy request, let p count the original prompt and any replay prefix, and let g count newly generated response tokens. The first output token is sampled from the prefill computation, leaving $m = \max(g-1, 0)$ incremental decoding forwards. Let $F(p, g)$ denote the decoding FLOP count for this request. The fixed coefficient A counts attention projections, the gated feed-forward layers, and vocabulary logits per decoding forward. The coefficient B counts attention matrix multiplications per context position per decoding forward. For the Qwen3-4B architecture used here, our matrix-multiplication accounting is

$$F(p, g) = Am + B\left(mp + \frac{m(m+1)}{2}\right), \qquad A = 8{,}044{,}544{,}000, \quad B = 589{,}824. \tag{7}$$

The architecture has 36 layers, hidden dimension 2,560, feed-forward dimension 9,728, 32 query heads, eight key/value heads, head dimension 128, and vocabulary size 151,936. A multiply and an addition count as two FLOPs. We exclude prefill, validation, value function and judge calls, training forward/backward computation, elementwise operations, communication, and failed attempts absent from completed-request logs. This accounting is not a hardware counter or a latency model; memory bandwidth can affect runtime without changing the arithmetic count.

Cumulative cost sums completed requests, including responses collected for replay refresh, without multiplying by node count. A branch inherits the parent's cumulative cost through its checkpoint. Table 4 lists the available cost-data horizons, which can extend beyond the last evaluation. Where paired per-request prompt-plus-prefix and generated lengths are unavailable, let N be the number of completed requests and let $\bar{m}$ and $\bar{p}$ be their empirical mean decoding-forward count and prompt-plus-prefix length. Let F_{total} be the sum of their per-request decoding costs. We estimate this total by substituting the means into the cost formula, giving the estimate $\widehat{F}$,

$$\widehat{F} = N\left\{A\bar{m} + B\left[\bar{m}\bar{p} + \frac{\bar{m}^2 + \bar{m}}{2}\right]\right\}. \tag{8}$$

For empirical moments defined with denominator N, the omitted term is

$$F_{\text{total}} - \widehat{F} = NB\left[\operatorname{Cov}(m, p) + \tfrac{1}{2}\operatorname{Var}(m)\right]. \tag{9}$$

Consequently, the estimate need not be a lower bound. The per-step approximation uses total generated tokens divided by request count and weights replay-only prefix means by the fraction of replay requests. It assumes each request generates at least one token and the logged prompt mean represents all requests. All GRPO curves use this mean-length estimate, with original training statistics covering steps 1–70, 1–180, and 1–100 for learning rates 10^{-6}, 2×10^{-6}, and 4×10^{-6}, respectively. For the selected GRPO run, this approximation underestimates the cost of the retained complete records at steps 161–180 by 4.01%. Other runs use original joint length records in preference to means where available. Applying a mean-only calculation to the main run's complete length records underestimates its cost by 5.51%; this is a descriptive check on that run, not an error bound for the missing data. We apply no universal correction factor.

Table 4: Available decoding-cost records. Joint lengths retain per-request prefix/decode moments. Step means yield estimated costs. The last cost-data step can exceed the last evaluation.

Configuration	Length source	Last cost step
AC2	Joint lengths	202
GRPO, 10^{-6}	Step means	70
GRPO, 2×10^{-6}	Step means	180
GRPO, 4×10^{-6}	Step means	100
Prefix GRPO	Means + joint lengths	120
No audit, branch at 40	Joint lengths	123
No audit, from initialization	Means + joint lengths	173
AC2 w/ correct-only buffer	Means + joint lengths	117
AC2 w/o Group & Audit (step-50 branch)	Joint lengths	117
AC2 w/o Group & Audit (from base model)	Step means	186
AC2 w/o Group & Audit & local readiness (step-20 branch)	Joint lengths	40
AC2 w/ 2k chunks	Step means	128
AC2 w/ stale replay buffer	Means + joint lengths	146
75k response limit	Joint lengths	161

Relation to GPU hours. We compare Decoding FLOPs with GPU hours for the main AC2 run. For each training step, let D denote its policy-rollout decoding cost from Equation 7, summed over requests, and let H denote its GPU hours: the logged training-step duration in hours multiplied by 32 GPUs. The timer includes generation, value inference, judging, parameter updates, and checkpointing, but excludes validation, startup, and failed attempts. We use 196 complete steps from steps 1–202, excluding six steps resumed from cached intermediate results because their timers omit earlier computation.

We fit H as a linear function of D by ordinary least squares with an intercept. The predicted GPU hours, $\widehat{H}$, are $\widehat{H} = 6.11 + 27.41\,(D/10^{18})$. The per-step Pearson correlation is $r = 0.874$, with $R^2 = 0.763$ (Figure 22). This association supports Decoding FLOPs as a proxy for training cost in this run. We regress individual steps rather than cumulative totals to avoid correlation induced by accumulation.

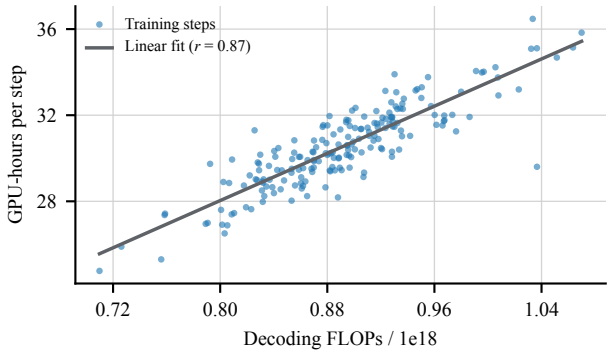

Figure 22: **Decoding FLOPs correlate with training-step GPU hours.** Each point represents one complete training step of the main AC2 run; the line is the least-squares fit. Both axes show per-step quantities.

A.6 TRAINING AND EVALUATION IMPLEMENTATION

Table 5: Main AC2 configuration and evaluation settings. Symbols follow Section 3. The total response budget includes replayed prefixes; b counts new tokens. Settings without a symbol in Section 3 are listed by name.

Parameter	Symbol	Value
Group size	g	16 continuations per prefix
Audit fraction	—	$1/4$; eligible-group count rounded up
Problems sampled for policy training per step	n_{batch}	192
Fresh inflow per step	n_{refill}	192 problems, one response each
Replay capacity	—	256 attempts
Total response budget	—	50,000 tokens
Short continuation budget	b	10,000 new tokens
Prefix cut grid / maximum fraction	—	10,000 tokens / 0.9
Value classes	—	$\{0, 0.1, \ldots, 1\}$
Value buffer capacity	—	1,920 records
Value records per update	—	Up to 768
Valid scores per grouped value target	—	At least 8
Policy learning rate	—	2×10^{-6}
Initial value learning rate	—	$2\sqrt{2} \times 10^{-6}$
Value learning-rate floor	—	$5\sqrt{2} \times 10^{-7}$
Policy minibatches	—	1,536 continuations each
Value gradient norm clip	—	0.2
Value decoding	—	Greedy; at most four tokens
Readiness window	—	5 training steps
Pooled MAE threshold	τ_{global}	0.20
Per-problem error threshold	τ_{local}	0.18
Training temperature / top-p / top-k	—	0.8 / 1 / unrestricted
Evaluation responses per problem	—	16
Evaluation interval	—	10 outer training steps
Evaluation temperature / top-p / top-k	—	0.8 / 0.95 / 20
Evaluation response limit	—	50,000 tokens

Replay construction. The policy, replay buffer, value training buffer, and reference bank start from the base model and empty data stores. While replay is empty, we generate responses to fresh problems with empty response prefixes for policy training. Once replay is available, we randomly permute its distinct problems and cycle through this order until n_{batch} problems have been sampled, allowing repetitions. For each sampled problem, we uniformly select one stored attempt. Fresh inflow attempts are admitted regardless of judged success. For an attempt of length L, the prefix length is drawn uniformly from multiples of 10,000 tokens in $[0, 0.9L]$; zero means starting from the question. Replay retains up to $C_R = 256$ attempts with global first-in, first-out replacement. The separate inflow of $n_{\text{refill}} = 192$ fresh problems produces one full-budget response per problem and is excluded from the policy loss. The total response limit counts both the stored prefix and newly generated tokens.

Reference bank maintenance. The reference proof bank $\mathcal{B}_{\text{ref}}$ persists independently of the replay FIFO. Both replay continuations and fresh responses can supply reference proofs. A response must receive a valid score of at least six out of seven judge points. For each previously uncovered problem, we select one eligible response using the seeded bank-selection RNG and store its extracted proof. Existing bank entries are retained. The readiness update uses the bank membership before these insertions; the subsequent value-training draw can use the new references.

Value examples and reference conditioning. We admit a grouped prefix's value target when it has at least $m_{\min} = 8$ valid scores; groups with one continuation use threshold one. Invalid prefix-value baselines in these groups exclude the corresponding row from both value targets and policy loss. Nearest-grid rounding breaks ties toward the smaller value. Each value update samples up to $N_V = 768$ fitting records from the buffer of capacity $C_V = 1{,}920$. We uniformly permute the records and take the first eligible records that fit the context limit, without replacement. A record retains its source-trajectory reference when available. Otherwise, the current reference bank supplies a fallback at sampling time, so an older record may acquire reference conditioning on a later draw. When a judged-correct reference is available, a record supplies prompts with and without that reference, each with weight $1/2$. Without a reference, it supplies only the no-reference prompt with weight one. References require at least six of seven judge points. The prompt asks for expected normalized rubric credit within the remaining response budget. Supervision covers the numeric class label and end-of-turn marker. The log-likelihood sums over these tokens. The implementation multiplies the weighted negative log-likelihood of Section 3.3 by $1/4$, using a fixed divisor rather than the label length. Inference greedily decodes at most four tokens and parses a leading legal grid value. This procedure does not compute a probability-weighted average of the eleven classes.

Readiness and auditing. Fresh probes compare pre-update value predictions with newly collected, rounded group targets; invalid predictions receive error one. Fresh-probe MAE is the mean absolute error over these probes. A problem becomes eligible for short continuations when the pooled fresh-probe MAE over five complete steps is below 0.20 and its own fresh prediction error is below 0.18. It must also have a nonzero value prediction observed in the current or an earlier iteration and a judged-correct reference in the bank. Eligibility persists after these conditions are met. These targets can include value substitutions, so eligibility is not a certificate of accuracy against independent terminal rewards. Let N_{ready} be the number of sampled ready problems at a step. With audit fraction $\alpha = 1/4$, we uniformly select $\lceil \alpha N_{\text{ready}} \rceil$ of these problems for full groups. The remaining sampled ready problems receive short groups; sampled unready problems always receive full groups. Short groups use a budget clipped to the remaining response window. The value function replaces the terminal reward only when this short budget is reached before the global limit without an extractable proof. Natural completion, global-limit exhaustion, and extractable proofs use judging.

Scoring failures. Missing proof content is a valid zero reward. Failed judge requests or unparsable judge responses are excluded from value targets but remain zero-reward rows in the policy update. Invalid consumed-value predictions receive zero policy-loss weight; their reward entries are filled with the mean of valid sibling scores, or zero when none exist, before computing the baseline over the full group. When all scores are valid, each continuation's advantage is its score minus the group mean. A group without valid scores supplies no value target and no nonzero policy advantage.

Shared-parameter optimization. Each outer step applies the first PPO minibatch update, one value classification update, and then the second PPO minibatch update. Each policy minibatch has 1,536 continuations. Sequence-length balancing first reorders individual rows. Group advantages are then computed by group identifier on the full training batch, before rows are partitioned into optimizer minibatches. Members of a rollout group may therefore contribute to both policy updates. Policy loss is averaged over unmasked generated tokens in each global minibatch. Training uses one policy epoch with explicit minibatch shuffling disabled. The value update remains applied to the shared parameters; optimizer states are separate. Its AdamW optimizer uses betas $(0.9, 0.999)$, epsilon 10^{-8}, and no weight decay. Let $\Delta\theta_V$, $\Delta\theta_{\text{PPO},1}$, and $\Delta\theta_{\text{PPO},2}$ be the parameter changes from the value update and the first and second policy updates, respectively, each measured from the parameters immediately before that update. The value learning rate η_V, initialized at $2\sqrt{2} \times 10^{-6}$ (Table 5), is halved after two consecutive steps in which $\|\Delta\theta_V\|_2 / (\|\Delta\theta_{\text{PPO},1}\|_2 + \|\Delta\theta_{\text{PPO},2}\|_2) > \sqrt{2}$, subject to the floor in Table 5. Policy optimization uses gradient clipping at 0.3 and weight decay 0.01, with no KL penalty or additive entropy bonus. The dual-clip coefficient is 3.

Adaptive entropy control. Let $\widehat{H}$ be the mean categorical entropy in nats under the pre-update policy at the rollout temperature. We average over generated-token positions in responses with an empty replay prefix, including fresh responses and zero-length replay cuts. Nonempty-prefix continuations are excluded from this control signal. Given target entropy $H^\star$, step size δ_H, a persistent

offset k bounded by $k_{\min}$ and $k_{\max}$, and base upper margin $\epsilon_{\text{high}}^{\text{base}}$, each iteration updates

$$\begin{aligned} k &\leftarrow \text{clip}\Big(k + \delta_H \,\text{sign}(H^\star - \widehat{H}), k_{\min}, k_{\max}\Big)\,, \\ \epsilon_{\text{high}} &= \epsilon_{\text{high}}^{\text{base}} + k. \end{aligned} \tag{10}$$

The controller is updated once before policy optimization; the resulting upper margin is fixed across both policy minibatches. The lower margin ϵ_{low} and base upper margin remain fixed. We use $\epsilon_{\text{low}} = 0.2$, $\epsilon_{\text{high}}^{\text{base}} = 0.28$, $H^\star = 0.28$, $\delta_H = 0.02$, $k_{\min} = -0.08$, $k_{\max} = 0.08$, and initialize $k = 0.06$. The offset persists across training iterations and checkpoint resumes.

Evaluation and baseline. Evaluation draws 16 responses per problem every ten outer training steps. The main comparison uses a 50,000-token evaluation response limit. GRPO samples 256 fresh problems with 16 responses per problem at each training step, whereas AC2 trains on the $n_{\text{batch}} = 192$ groups in Table 5. Equal Steps therefore need not imply equal numbers of responses or equal decoding work. The GRPO sweep uses learning rates 10^{-6}, 2×10^{-6}, and 4×10^{-6}; its observed durations and selection procedure are reported in Appendix A.1.

B Extended Related Work

AC2 is an actor-critic method as it learns a value model and policy, improving the policy using the value model (Mnih et al., 2016; Fujimoto et al., 2018; Haarnoja et al., 2018). Astute readers will have noticed that our method name is very similar to A3C (Mnih et al., 2016), and this reflects the similarity in the methods themselves (although the application settings are vastly different). A3C estimates the advantage from an n-step return with n far smaller than the episode horizon, using the critic at step n. It then uses this advantage to improve the actor through a policy gradient update. Our use of the GAE $\lambda = 0$ advantage estimator on critic ready problems to get the advantage of a *single* action chunk without rolling out to the end of the trajectory is similar in nature.

Despite their success in continuous control, actor-critic methods have not been widely adopted for LLM RLVR. Instead, group-relative methods dominate (specifically GRPO (Shao et al., 2024) and its descendants (Yu et al., 2025; Chen et al., 2025; Khatri et al., 2026)). These methods sample a group of complete responses per problem and use the group mean of the terminal rewards as a Monte-Carlo baseline. In the language of §2 this is the $\lambda = 1$ estimator with the baseline evaluated at the root. This *requires* every rollout to run to termination, and every token of a response receives the same advantage.

In response to the latter concern, a recent wave of work reintroduces a learned value model to RLVR, but keeps it in the role of a baseline. Le Critique (Venkatraman et al., 2026), EVPO (Pan et al., 2026), BPCO (Qi et al., 2026), POISE (Choi et al., 2026), $V_{0.5}$ (Zhang et al., 2026), GenAC (Shan et al., 2026), and VAPO (Yue et al., 2025) each train a critic alongside the policy and subtract it when forming advantages. One can also solve the uniform advantage problem without a learnt value model, by estimating intermediate values with additional Monte-Carlo rollouts to terminal reward. This is the approach taken by VinePPO (Kazemnejad et al., 2025) and SPO (Guo et al., 2025); however, this requires generating a large number of tokens for each advantage calculated. As the value estimate (whether from a value model or additional rollouts) is used (primarily) for baseline only, all these methods sit at or near $\lambda = 1$ in Equation 3. Note any $\lambda > 0$ needs to observe the terminal reward, and to our knowledge AC2 is the first method to date to use $\lambda = 0$ for LLM RLVR.

Using the value model so aggressively *requires* an accurate critic. We achieve this through (1) local critic readiness, and (2) providing the critic with privileged information (a correct solution) if we have it. Both have precedent in the literature. Conditioning the critic on privileged information is also done by Le Critique and BPCO (Venkatraman et al., 2026; Qi et al., 2026). Parameterising the critic as the language model itself, rather than a scalar head, follows GenAC (Shan et al., 2026). Deciding *when* to trust the critic has been posed globally, by switching or interpolating baselines on a reliability signal (Pan et al., 2026; Zhang et al., 2026; Venkatraman et al., 2026). We demonstrate that local is also very important (Figure 3).