BEYOND FIXED GROUPS: ADAPTIVE ROLLOUTS FOR CODING-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon coding-agent rollouts have highly variable latency: an agent may search a repository, edit files, and run several test suites before receiving one verifiable reward. Fixed-size GRPO allocates the same number of rollouts to every query, while round-based collection delays resampling and fresh work behind stragglers. We present *ARCA*, an event-driven realization of Adaptive Rollouts for Coding Agents. It processes each query when its current sampling stage completes, closes mixed groups, resamples all-wrong groups within a bounded budget, and refills fresh queries. The policy stays fixed during collection; the learner updates after collection and draining. In single-run experiments with Qwen3.6-27B on SWE-bench Verified, the $(8, 6, 4)$ schedule at fresh-query width $b = 16$ records 2.44 rollout steps/h versus 0.87 for native $N = 16$ GRPO ($2.80\times$). Observed peak pass@1 on the 51-problem evaluation slice is 0.608 versus 0.510, a gain of 9.8 percentage points. At $b = 8$, $(8, 6, 4)$ reaches 0.627 peak pass@1, while $(8, 4, 4)$ records the highest throughput, 3.04 steps/h. These throughput comparisons use different mixed-group targets and do not measure total compute savings.

1 MOTIVATION

Improving the speed while maintaining the training performance has always been a goal for agentic post-training, including the coding agent, where long-horizon coding-agent rollouts have highly variable latency: an agent may search a repository, edit files, and run several test suites before receiving one verifiable reward.

However, the current GRPO has a potential limitation that uses a fixed size of group (eg. N=16) for every query, regardless of the queries' difficulty. It wastes time on rollouting the large amounts of queries that are homogeneous. Insufficient rollouts are common for difficult queries. Wasted rollouts are given for easy queries eventually dropped. At the same time, the inhomogeneous queries do not require the full fixed group size to be mixed for trainer. Thus, in this work, we provide a strategy that dynamically buckets the queries into groups based on their difficulty status to improve the efficiency.

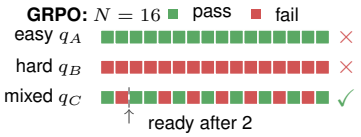


Figure 1: Homogeneous groups have zero advantage.

Adaptive methods already show that rollout budgets need not be fixed per query (Zhang et al., 2026; 2025; Nomand et al., 2026b). NGU also resamples asynchronously (Noukhovitch et al., 2026). In coding-agent training, rollout times vary greatly: a short edit may finish quickly, while another trajectory waits for repository setup, tool calls, or tests. Under round-based collection, completed queries cannot be processed until the synchronization barrier, delaying resampling and fresh-query dispatch. The problem is therefore not only how many rollouts to allocate to each query, but also when to close, resample, or replace a query as results arrive.

Our work makes three contributions: 1. Our work produces more effective rollouts from difficult queries, and it also solves the problem of generating effective training samples, more efficient than the GRPO recipe. 2. Our work produces a rollout allocation strategy that integrates into the current agentic coding agent training framework, becomes a standard prototype. 3. At $b = 16$, ARCA-$(8, 6, 4)$ records $2.80\times$ the rollout-step throughput of native $N = 16$ GRPO and a 9.8-percentage-point higher observed peak pass@1 on the SWE-bench Verified evaluation slice.

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.

2 Related Work

GRPO uses within-group reward differences (Shao et al., 2024), while DAPO replaces homogeneous groups (Yu et al., 2025). Prompt selection draws on reward history in GRESO (Zheng et al., 2025b), difficulty predicted from a reference set in DOTS (Sun et al., 2025), and observed or predicted reward variability in VCRL (Jiang et al., 2025) and Goldilocks (Mahrooghi et al., 2026). HeaPA samples and augments prompts by difficulty (Wang et al., 2026b); CoDaPO uses confidence and difficulty to reweight and resample questions (Zhou et al., 2026), and TAROT tailors test curricula to initial coding capability (Park et al., 2026). Per-prompt budgets use initial-policy performance in sGPO (Sudalairaj et al., 2026), predicted success rates in VIP (Nguyen et al., 2026), or pilot outcomes in HORA (Wang et al., 2026a). DARS expands sampling for low-accuracy problems (Yang et al., 2025), while GDRO studies prompt reweighting separately from rollout allocation under a fixed mean budget (Panaganti et al., 2026). AR3PO samples in stages and can reuse cached successes (Zhang et al., 2025); SARA makes posterior-based accept, abandon, or continue decisions in synchronized rounds (Nomand et al., 2026b). NGU asynchronously resamples all-wrong prompts with probabilistic stopping and uses past rewards to adjust advantages (Noukhovitch et al., 2026); ARCA instead caps rollouts per query and keeps the policy fixed during collection.

TRACE predicts success at prompt roots and intermediate histories to allocate full rollouts for mixed outcomes and prefix continuations for outcomes that differ from the original rollout (Zou et al., 2026). PAIR selects continuations by predicted correctness and token cost, and corrects pairwise gradient terms using joint inclusion probabilities (Nomand et al., 2026a), while Prefix Sampling replays successful or failing prefixes to balance pass rates (Zhu et al., 2026). The original AERO combines staged sampling with Bayesian advantages for homogeneous groups and failure downsampling (Zhang et al., 2026). Our coding-agent variant retains initial sampling and resampling but omits both adjustments. It trains on observed mixed groups, drops all-correct initial groups, bounds resampling for all-wrong groups, and refills fresh queries as groups resolve. We allocate complete episodes without learned predictors or prefix replay and compare against fixed-group GRPO.

3 Adaptive Rollout

At learner step t, all rollouts use the fixed policy $\pi_t = \pi_{\theta_t}$. For query q, rollout j has binary reward $r_{q,j}$. Let $G_q^{(k)}$ contain the indices of its rollouts after shot k, where shot 0 is initial sampling and later shots are resampling stages. Here resampling generates additional rollouts for the same query and appends them to its existing group; it does not redraw or discard the earlier rollouts. A pool is mixed when

$$0 < \sum_{j \in G_q^{(k)}} r_{q,j} < |G_q^{(k)}|. \tag{1}$$

Thus the pool contains at least one pass and one failure.

The learning step for agentic training differs from that in the single-turn RLVR setting: rollouts are scheduled asynchronously and use a refill mechanism. Figure 2 shows this event-driven schedule with native GRPO. In the rollout stage, agents interact with the environment hosted by a sandbox to complete the task. The rollout process is not only a model call but also a runtime for an agent with tools, MCPs, and models. The bottleneck for agent post-training comes from this complicated rollout stage. To conquer this difficulty, the rollout process has been changed in several ways. First, queries are scheduled asynchronously. The sampler processes each rollout as soon as it finishes, rather than waiting for the entire batch. Once all rollouts for that query's current sampling stage finish, the sampler decides whether to accept, resample, or drop the query. A homogeneous all-correct or all-incorrect initial group is dropped immediately after the final shot. Second, a refill mechanism is used once a dropped query leaves the live set. When the remaining-query count falls below the configured burst width, the sampler launches another fresh-query burst, subject to an in-flight guard.

The difficult queries are more important for agentic training. Agents get better skills by learning from solving difficult tasks. However, even in agentic training, the sampler deals with all-correct and all-incorrect queries in the same way. The model should learn from the difficult ones. This also causes inefficiency, as all the queries enter the rollout engine with a uniform group size, with no difference between difficult and easy ones.

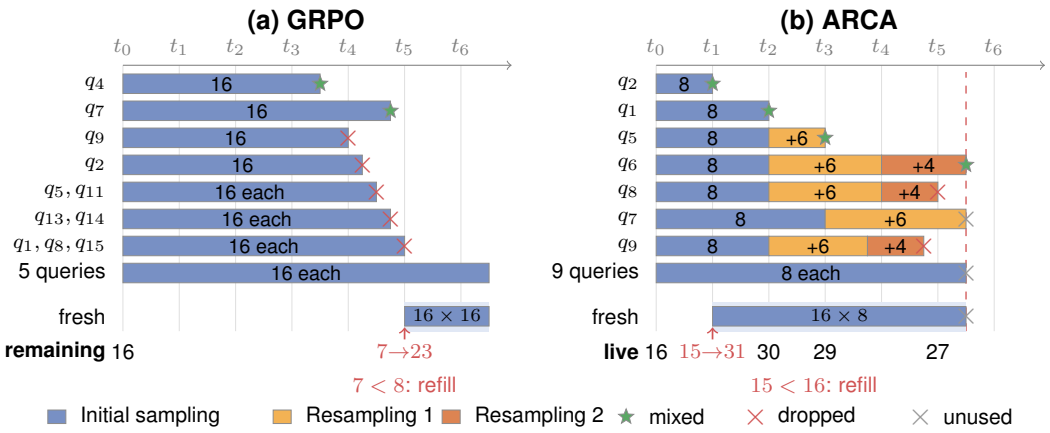


Figure 2: Query schedules: native GRPO (left) refills after enough discards; ARCA (right) refills as queries close. All-wrong q_8 and q_9 drop after both resampling stages. Bar labels count rollouts; the dashed line ends dispatch before draining. Time is schematic, not measured.

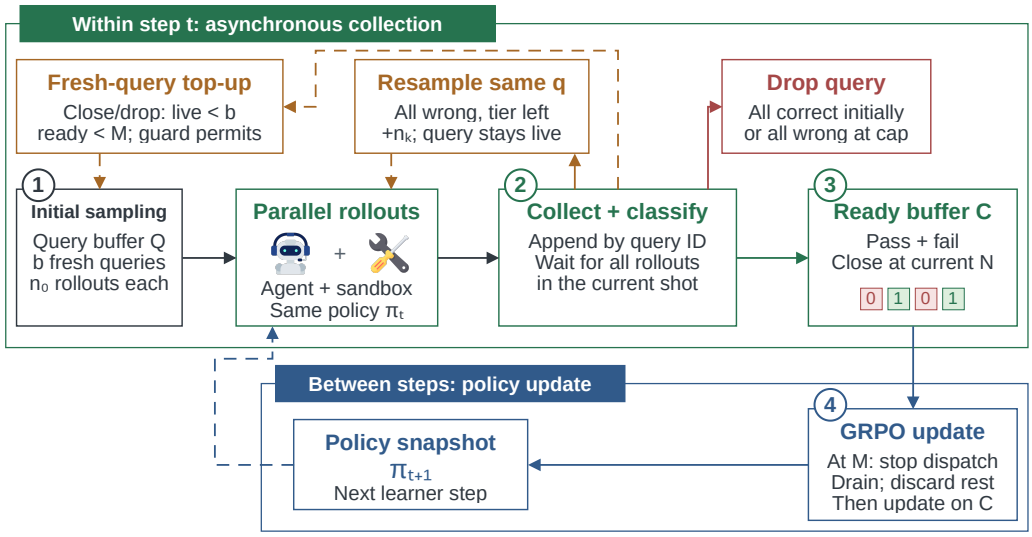


Figure 3: ARCA collects mixed groups through initial sampling, resampling, and early stopping, then updates the policy after collection and draining.

Our method is designed to improve the efficiency of generating effective training samples. We call it adaptive sampling. Figure 3 summarizes our method for a whole training step. **Asynchronous rollout allocation.** Once a query has both correct and incorrect rollouts, sampling stops and the group is added to the training buffer. A homogeneous all-correct initial group is dropped immediately after the final shot. And a homogeneous all-incorrect group enters a resampling stage. **Fresh-query top-up.** A resampled query remains live and does not itself release a fresh-query slot. Only a query that closes as mixed enters the trainer. This preserves query-level grouping while avoiding a global straggler barrier.

Figure 2 shows our method in detail compared to GRPO. We provide the **details of adaptive resampling.** A fresh query receives n_0 initial rollouts. A mixed group closes immediately. In the drop-all-correct variant studied here, an all-correct initial group is discarded. An all-wrong group receives n_1 additional rollouts in Resampling 1; if it remains all-wrong, Resampling 2 adds n_2 more. We denote the sampling schedule by $\mathbf{n} = (n_0, \ldots, n_K)$, with $K = 2$ resampling stages in the tested configurations. Write $S_k = \sum_{\ell=0}^{k} n_\ell$ for the cumulative size after shot k. The $(8, 6, 4)$ schedule gives $S_0, S_1, S_2 = 8, 14, 18$. Throughout the paper, N denotes a group size: GRPO fixes it upfront, while ARCA closes at its current size $N = S_k$. Algorithm 1 presents our algorithm in detail.

Why adaptive sampling can save work. ARCA closes a mixed group or drops an all-correct group after eight rollouts; GRPO $N = 16$ waits for all 16. All-wrong queries can take longer under ARCA because resampling stages run sequentially, while GRPO's 16 rollouts are already in flight. If resampling finds a success, ARCA trains on the accumulated pool, including the initial eight failures. An all-wrong GRPO group is discarded at its cutoff. We implement these decisions in SLIME's `RolloutManager`, which submits trajectories to SGLang and tracks each query's rewards.

Learner update. ARCA holds π_t fixed while query stages run concurrently. Once the ready buffer reaches its target, such as four mixed groups, it stops dispatch, drains outstanding requests, discards unresolved groups, and updates on the closed pools. For a closed pool G_q, let $\bar{r}_q$ and σ_q be its reward mean and standard deviation. Since every retained rollout was sampled from π_t, the GRPO quantities are

Algorithm 1 ARCA with drop-all-correct at learner step t

Require: query stream $\mathcal{Q}$; fixed policy π_t; burst width b; schedule $\mathbf{n} = (n_0, n_1, n_2)$; ready target M
Ensure: learner-ready query-pool pairs $\mathcal{C}$
$\mathcal{O}, \mathcal{C} \leftarrow \varnothing$; initialize fresh pools and launch b queries with n_0 rollouts each; add their ids to $\mathcal{O}$
while $|\mathcal{C}| < M$ **do**
  Receive rollout (q, j); record $r_{q,j}$ and add j to G_q
  if the current shot for q is complete **then**
    if G_q has both a pass and a failure **then**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{(q, G_q)\}$; remove q from $\mathcal{O}$ {mixed: close}
    else if G_q is all-correct **then**
      drop G_q; remove q from $\mathcal{O}$ {all-correct: drop}
    else if a resampling stage k remains **then**
      launch n_k additional rollouts for q {all-wrong: resample}
    else
      drop G_q; remove q from $\mathcal{O}$ {exhausted: discard}
    end if
  end if
  if $|\mathcal{C}| < M$, $|\mathcal{O}| < b$, and the in-flight guard permits **then**
    Initialize b fresh pools, launch n_0 rollouts per query, and add their ids to $\mathcal{O}$ {top-up}
  end if
end while
Drain outstanding requests; discard unresolved groups; update on $\mathcal{C}$ with GRPO; sync $\pi_t \to \pi_{t+1}$

$$\widehat{A}_{q,j} = \frac{r_{q,j} - \bar{r}_q}{\sigma_q + \epsilon}, \quad \rho_{q,j,\tau} = \frac{\pi_\theta(a_{q,j,\tau} \mid s_{q,j,\tau})}{\pi_t(a_{q,j,\tau} \mid s_{q,j,\tau})}. \tag{2}$$

Here τ indexes tokens, $s_{q,j,\tau}$ is the generation history, and $\epsilon > 0$ stabilizes the denominator. Homogeneous groups have zero group-relative advantages. Mixed groups have positive and negative coefficients, although their presence alone does not guarantee a larger policy improvement.

4 Rollout Allocation Analysis

ARCA retains earlier outcomes and stops sampling a query once its group contains both correct and incorrect rollouts. For a fixed model and sampling schedule, we analyze the probability of forming such a group and the number of completed rollouts required.

Model and comparison. Let Y_q indicate whether query q becomes mixed before its stopping rule ends, and let C_q count its complete rollouts.

Assumption 1 (Within-query sampling)**.** Fix the behavior policy π_t and sampling settings. For query q, the potential rewards $R_{q,j}$ are conditionally independent $\mathrm{Bernoulli}(p_q)$ variables. The deterministic schedule $\mathbf{n} = (n_0, \ldots, n_K)$ has positive integer shot sizes and $n_0 \geq 2$. Each query runs to its own close or drop, without interruption by the global ready target. One complete rollout costs one unit.

Here $R_{q,j}$ is random and $r_{q,j}$ is its observed value. The implementation uses observed rewards; p_q is an analysis parameter. We write $p = p_q$ when considering one query.

Lemma 1 (Retaining a homogeneous pool)**.** *Under Assumption 1, N fresh rollouts are mixed with probability*

$$m_N(p) = 1 - p^N - (1-p)^N, \qquad N \geq 1. \tag{3}$$

Suppose a retained pool has only reward $r \in \{0, 1\}$, on a history of positive probability. With $n \geq 1$ additional rollouts, its probability of becoming mixed is

$$h_r(n, p) = \begin{cases} 1 - (1-p)^n, & r = 0, \\ 1 - p^n, & r = 1. \end{cases} \tag{4}$$

Compared with discarding that pool and drawing a fresh group of size n from the same query, retaining it increases the mixed probability by p^n for $r = 0$ and $(1-p)^n$ for $r = 1$.

The retained failures mean that resampling needs only one success; retained successes need only one failure. For $N \geq 2$, $m_N(p)$ is largest at $p = 1/2$; this refers to a query's success probability, not to a fraction of queries.

Proposition 1 (Early stopping at the same cap)**.** *Consider a staged reference rule that resamples both homogeneous outcomes and has cap $S_K = N$. For the same potential reward sequence, let $(Y_q^{\text{both}}, C_q^{\text{both}})$ be its mixed indicator and cost, and let Y_q^N be the fixed-N mixed indicator. For every sequence,*

$$Y_q^{\text{both}} = Y_q^N, \qquad C_q^{\text{both}} \leq N. \tag{5}$$

Consequently, processing the same query stream one query at a time reaches the first M accepted query instances with no more rollouts, whenever they exist.

Once a pool is mixed, more samples cannot change that status. The proposition therefore needs no independence assumption or knowledge of query difficulty. It applies, for example, to the $(8, 4, 4)$ staged reference rule versus fixed-16 sampling, before adding concurrent top-ups or a global stopping target. At $p = 1/2$, both have mixed probability 0.999969, while the staged rule uses 8.033 expected rollouts instead of 16.

What early all-correct drop changes. For the current rule, define $a_{\mathbf{n}}(p) = \mathbb{E}[Y_q \mid p_q = p]$ and $c_{\mathbf{n}}(p) = \mathbb{E}[C_q \mid p_q = p]$, where $a_{\mathbf{n}}(p)$ is the mixed rate with per-rollout success probability p, and $c_{\mathbf{n}}(p)$ is the expectation of spending rollouts. Then

$$a_{\mathbf{n}}(p) = 1 - p^{n_0} - (1-p)^{S_K}, \quad c_{\mathbf{n}}(p) = n_0 + \sum_{k=1}^{K} n_k (1-p)^{S_{k-1}}. \tag{6}$$

Relative to resampling both outcomes at the same cap, early all-correct drop saves $\sum_{k=1}^{K} n_k p^{S_{k-1}}$ expected rollouts and loses $p^{n_0} - p^{S_K}$ mixed-group probability. Thus it changes both cost and acceptance. The proofs are in Appendix A.2.

Proposition 2 (Mixed groups per rollout at a fixed success rate)**.** *Under Assumption 1, suppose $1 \leq n_k \leq n_0$ for each resampling stage and $N \geq n_0$. For every $p \in [0, 1]$,*

$$\frac{a_{\mathbf{n}}(p)}{c_{\mathbf{n}}(p)} \geq \frac{m_{n_0}(p)}{n_0} \geq \frac{m_N(p)}{N}. \tag{7}$$

The first inequality is strict when $K \geq 1$ and $0 < p < 1$.

All three tested schedules satisfy $1 \leq n_k \leq n_0$. At a fixed success probability p, these resampling stages increase mixed groups per rollout, $a_{\mathbf{n}}(p)/c_{\mathbf{n}}(p)$, relative to initial sampling alone. Larger stages are more likely to rescue a query in one shot. Across different query difficulties, the total ratio also depends on which queries consume extra shots. Appendix A.3 gives the condition for reducing the expected budget for M groups and a counterexample when caps differ.

When per-query dispatch saves time. Per-query dispatch closes, drops, or resamples a query as soon as its current stage finishes; round-based dispatch waits for every query's current stage. Proposition 3 compares these rules when both execute the same sampling stages. It does not compare ARCA with fixed-size GRPO, which launches all N rollouts at once. The experiments below compare full-step throughput and pass@1.

5 Experiments

Setup. We train Qwen3.6-27B with SGLang and Megatron on SWE-bench Verified (449 training problems and a 51-problem evaluation slice) using two nodes with 16 H200 GPUs (TP4/PP2/CP2) and global batch size 4. Curves show raw pass@1 faintly and a three-point centered mean in bold. Each curve is one run; reported maxima are observed mid-training peaks rather than multi-seed estimates. The full configuration is in Appendix A.1 (Table A1).

Table 1 groups results by fresh-query width ($b = 8$ or $b = 16$). Throughput (Steps/h) is the last logged rollout-step counter divided by recorded runtime; efficiency ratios use native GRPO job 2155

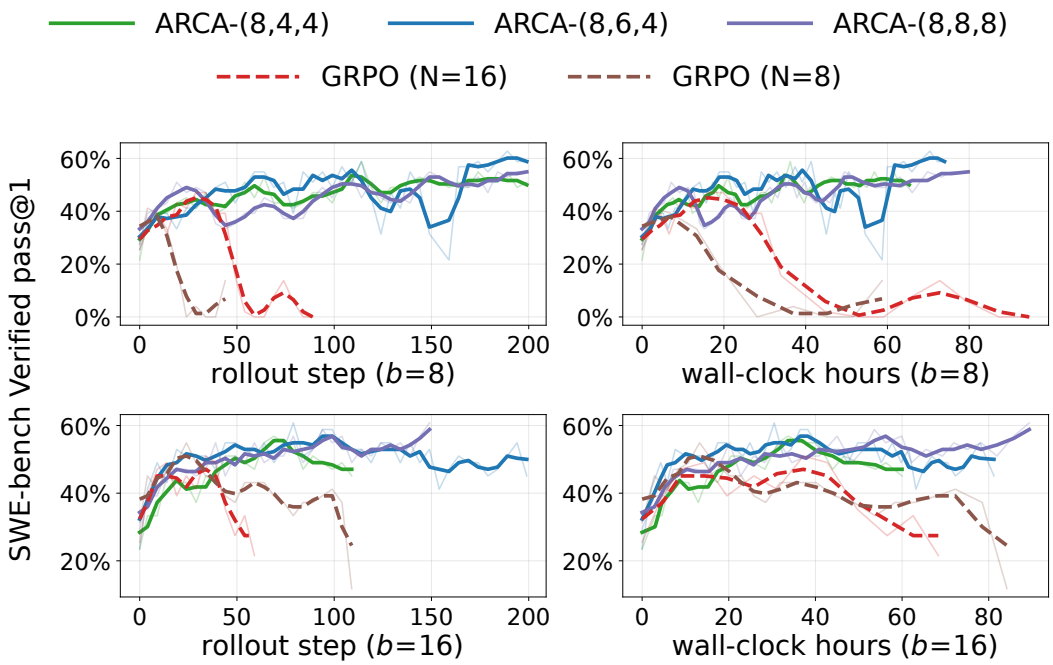


Figure 4: ARCA sampling schedules compared with fixed-size GRPO at two fresh-query wave widths. Top row: b=8, against rollout step and wall-clock time. Bottom row: b=16, same two views. Faint curves are evaluations; bold curves are centered three-point means; one shared legend (ARCA schedules, then GRPO group sizes) applies to all four panels.

as the reference. These comparisons cover the full training configurations: ARCA targets four mixed groups per rollout step, whereas native GRPO targets eight.

Table 1: Training throughput and peak pass@1 on SWE-bench Verified. Efficiency ratios and percentage-point differences use the native GRPO reference. Peaks are observed checkpoint maxima.

Setting	Rollout	Hours	Steps/h	Efficiency	max pass@1	Δ vs. GRPO
$b = 8$ **fresh queries per wave**						
GRPO $N = 16$	89	96.0	0.93	1.07×	0.471 (24/51)	−3.9 ppt
GRPO $N = 8$	89	68.1	1.31	1.50×	0.490 (25/51)	−2.0 ppt
ARCA $(8, 4, 4)$	199	65.6	**3.04**	**3.49**×	0.588 (30/51)	+7.8 ppt
ARCA $(8, 6, 4)$	199	74.0	2.69	3.09×	**0.627 (32/51)**	**+11.7 ppt**
ARCA $(8, 8, 8)$	199	80.0	2.49	2.86×	0.549 (28/51)	+3.9 ppt
$b = 16$ **fresh queries per wave**						
GRPO $N = 16$	62	71.5	0.87	1.00×	0.510 (26/51)	—
GRPO $N = 8$	109	84.2	1.29	1.48×	0.549 (28/51)	+3.9 ppt
ARCA $(8, 6, 4)$	199	81.4	2.44	2.80×	0.608 (31/51)	+9.8 ppt
ARCA $(8, 4, 4)$	109	60.0	1.82	2.09×	0.569 (29/51)	+5.9 ppt
ARCA $(8, 8, 8)$	149	89.4	1.67	1.92×	0.608 (31/51)	+9.8 ppt

Finding 1: ARCA versus GRPO. At both fresh-query widths, ARCA reaches higher peak pass@1 and maintains stronger late-training performance than the plotted GRPO controls (Fig. 4). At $b = 8$, the $(8, 4, 4)$, $(8, 6, 4)$, and $(8, 8, 8)$ schedules peak at 0.588 $(30/51)$, 0.627 $(32/51)$, and 0.549 $(28/51)$, respectively, and all reach rollout step 199. The $(8, 6, 4)$ run recovers from a mid-run dip to its late peak and ends at 0.588. GRPO $N = 16$ peaks at 0.471 $(24/51)$ but ends at zero at step 89; the plotted $N = 8$ run peaks at 0.412 $(21/51)$ before declining and ends at step 44. At $b = 16$, ARCA-$(8, 6, 4)$ and the GRPO controls perform similarly for roughly the first 25 hours, after which ARCA reaches 0.608 $(31/51)$ and continues to step 199. GRPO $N = 16$ falls from a peak of 0.510 $(26/51)$ to 0.216 $(11/51)$, while $N = 8$ peaks at 0.549 $(28/51)$ before falling below 0.40. The incomplete $(8, 4, 4)$ and $(8, 8, 8)$ runs reach steps 109 and 149, with peaks of 0.569 $(29/51)$ and

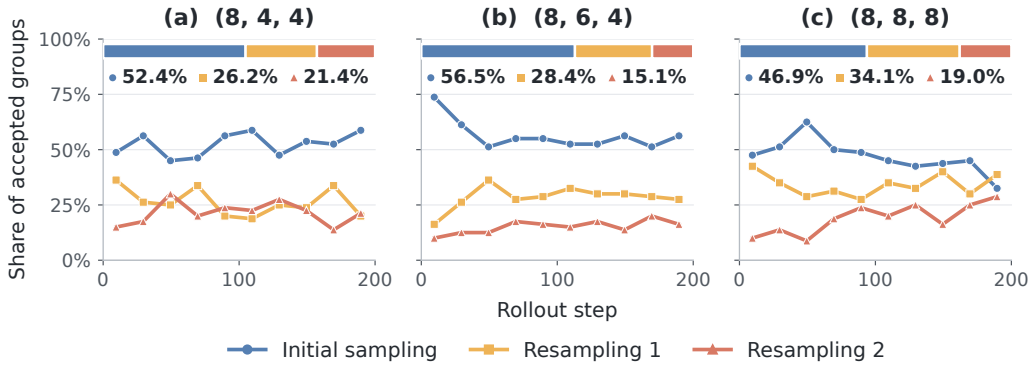


Figure 5: Closing stages of accepted mixed groups under three ARCA schedules. Bars show overall shares; curves show shares in 20-step bins. One 200-step run per schedule; homogeneous drops and unresolved work are excluded.

0.608 (31/51), respectively. Table 1 reports 3.04 steps/h for $(8, 4, 4)$ at $b = 8$ and 2.44 for $(8, 6, 4)$ at $b = 16$: 3.49× and 2.80× the native GRPO reference's 0.87 steps/h, respectively.

Finding 2: evidence of adaptive resampling stages. Figure 5 shows when accepted mixed groups close under the three sampling schedules. Each run spans 200 steps, we observe that effective queries needs different group size, 8, 12, 16, 18, 24. ARCA's adaptive sampling achieves this rollout allocation.

Finding 3: which instances are solved? We compare ARCA $(8, 6, 4)$ at $b = 16$ and GRPO $N = 8$ at their common final rollout step 199, without selecting peak checkpoints. Figure 6 groups the 51 evaluation tasks by the official SWE-bench Verified repair-time labels, independently of model outcomes.[1] ARCA solves 23 instances and GRPO 15: 14 are shared, nine are ARCA-only, and one is GRPO-only. Six of ARCA's nine exclusive successes are in the 15-minute–1-hour category. Neither method solves any of the six instances rated above one hour. Thus this comparison supports ARCA's effectiveness on solving more difficult tasks. Appendix A.5 reports earlier checkpoints and examines recorded edits, including a GRPO-only success.

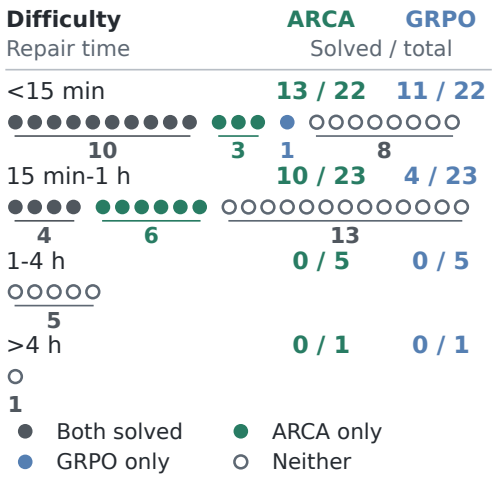


Figure 6: Paired outcomes at step 199 (one run pair). Dots denote tasks; counts appear below. Two shared execution or patch-application failures count as unsolved.

Finding 4: generalization beyond SWE-bench Verified. We evaluate two held-out benchmarks with no task overlap with the 449 training problems or the 51-problem checkpoint-selection split. SWE-Gym-100 contains 100 repository-repair tasks from 11 repositories absent from both splits. Each task uses one greedy, multi-turn attempt in a Docker sandbox, a 1,800 s agent budget, and binary test-based grading. Terminal-Bench-64 contains 64 tasks spanning 16 categories and tests broader terminal-agent capabilities beyond repository repair. We evaluate selected checkpoints from six ARCA and four GRPO configurations, plus each GRPO run's last saved checkpoint. Selection uses the maximum centered three-checkpoint mean pass@1 on the 51-problem split, without OOD feedback.

Table 2 shows that every selected ARCA checkpoint improves on the base model on both benchmarks. ARCA solves 18–26 SWE-Gym tasks, versus 12–18 for selected GRPO and eight for the base model; Terminal-Bench scores are 29.7%–50.0%, versus 25.0%–46.9% and 18.8%, respectively. Mean

[1]The `difficulty` field in the SWE-bench Verified dataset.

success is 21.5% versus 15.8% on SWE-Gym and 40.9% versus 35.9% on Terminal-Bench. Late GRPO means fall to 7.8% and 28.9%, showing that late-training degradation also extends to OOD tasks.

Table 2: OOD results. Selected checkpoints maximize the centered three-checkpoint mean pass@1 on the 51-problem split; late checkpoints are each GRPO run's last. Repeated (b, N) labels denote distinct configurations. Bottom rows average configurations equally (n), in percent.

Training configuration	Checkpoint	SWE-Gym	Terminal-Bench
Base model	–	8/100	12/64 (18.8%)
ARCA $(8, 4, 4)$, $b = 8$	selected	23/100	19/64 (29.7%)
ARCA $(8, 6, 4)$, $b = 8$	selected	21/100	30/64 (46.9%)
ARCA $(8, 8, 8)$, $b = 8$	selected	18/100	**32/64 (50.0%)**
ARCA $(8, 4, 4)$, $b = 16$	selected	**26/100**	23/64 (35.9%)
ARCA $(8, 6, 4)$, $b = 16$	selected	19/100	27/64 (42.2%)
ARCA $(8, 8, 8)$, $b = 16$	selected	22/100	26/64 (40.6%)
GRPO $b = 16$, $N = 8$	selected	12/100	16/64 (25.0%)
GRPO $b = 16$, $N = 8$	late	10/100	14/64 (21.9%)
GRPO $b = 16$, $N = 8$	selected	15/100	17/64 (26.6%)
GRPO $b = 16$, $N = 8$	late	5/100	17/64 (26.6%)
GRPO $b = 16$, $N = 16$	selected	18/100	29/64 (45.3%)
GRPO $b = 16$, $N = 16$	late	9/100	17/64 (26.6%)
GRPO $b = 16$, $N = 16$	selected	18/100	30/64 (46.9%)
GRPO $b = 16$, $N = 16$	late	7/100	26/64 (40.6%)
GRPO-late mean ($n = 4$)		7.8%	28.9%
GRPO-selected mean ($n = 4$)		15.8%	35.9%
ARCA-selected mean ($n = 6$)		21.5%	40.9%

Table 3: Terminal-Bench success by difficulty, averaged over the six ARCA and four GRPO selected configurations in Table 2. All configurations use the same tasks.

Difficulty (tasks)	Base model	ARCA mean	GRPO mean
Easy (20)	40.0%	65.0%	56.3%
Medium (29)	10.3%	36.2%	34.5%
Hard (15)	6.7%	17.8%	11.7%

Table 3 shows the largest relative ARCA advantage on hard tasks: 17.8% versus 11.7%, a 6.1-percentage-point gap or 52.4% relative improvement. The absolute gap is largest on easy tasks, and the relative advantage is not monotonic across difficulty tiers. These configuration averages suggest that ARCA's gains extend to harder OOD tasks, while the small hard subset (15 tasks) limits precision.

6 Training Stability

GRPO training repeatedly collapsed in our experiments; only one inspected run showed no collapse within its logged horizon. We examine two complementary diagnostics: mixed-group composition and repeated tool-call loops.

Mixed-group composition and collapse. Figure 7 tracks mixed, all-correct, and all-wrong groups alongside pass@1 for three GRPO runs. The baseline discards homogeneous groups, whose relative advantages vanish. Early improvements accompany a falling all-wrong share and a larger mixed share; the all-correct share tracks performance less consistently.

During collapse, the pattern reverses: in panel (a), the all-wrong share rises from 46% to 68% between steps 44 and 54, while pass@1 falls from 20/51 to 1/51. Crucially, the subsequent recovery in group composition does not restore performance: the all-wrong share falls back to about 47% at step 59, yet pass@1 falls further to 0/51. Panels (b) and (c) show similar shifts near the marked steps 19 and 29, respectively. These correlations neither establish the cause of collapse nor guarantee recovery when the mixed share rebounds. The full traces, including the single inspected run that shows no observed collapse, are in Appendix A.6.

Loops as a concrete failure mode. We flag an evaluation episode when at least four consecutive recorded tool calls share the same tool name and first-argument prefix (up to 300 characters). In an inspected failure, the agent correctly diagnoses a bug and repeatedly announces an edit, but keeps re-reading the same source line until its budget is exhausted (Appendix A.6).

Figure 8 compares the first two and last two logged evaluations on the fixed 51-problem split. Across 13 GRPO-family log segments, the flagged fraction rises from 2.3% to 37.5%, while solve rate falls from 32.3% to 10.5%: repetition becomes both more common and more strongly associated with failure. Across nine ARCA logs the flagged fraction rises more modestly, from 0.9% to 4.9%, while

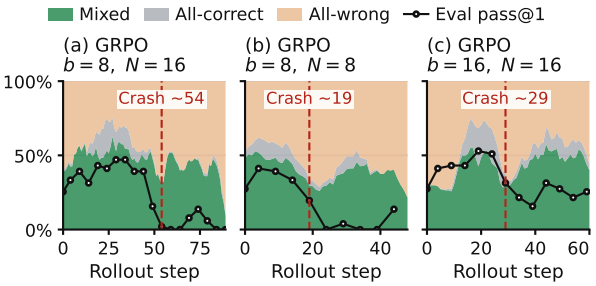


Figure 7: Sampled-group composition in GRPO: (a) $b = 8$, $N = 16$; (b) $b = 8$, $N = 8$; and (c) $b = 16$, $N = 16$. Stacked areas distinguish mixed, all-correct, and all-wrong groups (seven-point centered means); black curves show raw evaluation pass@1. Red dashed lines mark approximate declines, not detected change points.

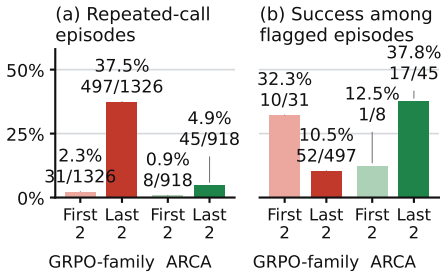


Figure 8: Loop diagnostics on the 51-problem split, pooling the first two and last two logged checkpoints per log segment. (a) Loop frequency: flagged/all episodes. (b) Loop solve rate: solved/flagged episodes. GRPO pools 13 log segments and ARCA nine logs; log segments are not necessarily independent runs.

solve rate increases from 12.5% to 37.8%. Repetition does not necessarily imply failure: in another GRPO run ($b = 16$, $N = 8$), 18 of 29 flagged episodes succeed at step 34.

When failed loops belong to an all-wrong group, GRPO has no reward contrast to penalize them. If ARCA resampling finds a success, the retained failures receive negative relative advantages. This can restore a corrective signal, but does not guarantee recovery. The logs differ in configuration and duration and repeatedly evaluate the same tasks, so they neither isolate the effect of resampling nor establish a cause of collapse.

Related observations include zero-advantage groups addressed by DAPO's dynamic sampling (Yu et al., 2025) and *zero-advantage lock-in* after format collapse in CIGPO's HotpotQA experiments (Dou, 2026). GSPO also reports GRPO collapses that are difficult to recover from, attributing them to importance-weight noise (Zheng et al., 2025a). These findings motivate complementary explanations; our group-composition and loop diagnostics do not establish a shared cause.

7 Conclusion and Limitations

ARCA uses initial sampling, resampling, and early exit for long-horizon coding agents. The reported runs achieve higher rollout-step throughput and observed peak pass@1 than fixed-N=16 and N=8 GRPO. OOD gains include hard tasks, although absolute gains are larger on easy tasks. The analysis identifies conditions for forming mixed groups with fewer rollouts.

Limitations. Each learner step drains outstanding requests and discards unresolved groups; cross-step reuse is not implemented and must account for policy changes. Schedule efficiency depends on query difficulty, generation and verification costs, and concurrency; choosing schedules for quality under a fixed compute budget remains future work. Experiments use one model, lack multi-seed estimates, and have small difficulty subsets; hard-task gains need confirmation on more tasks and repeated runs.

AI USE STATEMENT

Generative AI tools, including OpenAI Codex, assisted with implementation, code editing, literature search, bibliography organization, translation, prose editing, figure preparation, and the allocation analysis and its numerical checks. The authors take responsibility for the final code, text, references, proofs, claims, and results.

ETHICS STATEMENT

This work studies reinforcement learning for coding agents that edit repositories and execute commands in sandboxes. Such agents can produce incorrect or insecure code; deployment requires appropriate isolation, access controls, and human review. Our benchmark results measure task performance, not the safety of autonomous deployment.

REPRODUCIBILITY STATEMENT

Algorithm 1 gives the pseudocode for ARCA's sampling, resampling, and fresh-query top-up. Appendix A.1 (Table A1) specifies the model, hardware, software versions, optimization and decoding settings, and sampling schedules. Section 5 describes the data splits, checkpoint selection, and evaluation protocols; Appendices A.2–A.4 provide the proofs for the rollout allocation analysis.

REFERENCES

Hao Dou. CIGPO: Contextual information-gain policy optimization for multi-turn evidence-reading LLM agents, 2026. URL https://arxiv.org/abs/2607.16244.

Guochao Jiang, Wenfeng Feng, Guofeng Quan, Chuzhan Hao, Yuewei Zhang, Guohua Liu, and Hao Wang. VCRL: Variance-based Curriculum Reinforcement Learning for Large Language Models. *arXiv preprint arXiv:2509.19803*, 2025. URL https://arxiv.org/abs/2509.19803.

Ilia Mahrooghi, Aryo Lotfi, and Emmanuel Abbe. Goldilocks RL: Tuning Task Difficulty to Escape Sparse Rewards for Reasoning. *arXiv preprint arXiv:2602.14868*, 2026. URL https://arxiv.org/abs/2602.14868.

Hieu Trung Nguyen, Bao Nguyen, Wenao Ma, Yuzhi Zhao, Ruifeng She, and Viet Anh Nguyen. Adaptive Rollout Allocation for Online Reinforcement Learning with Verifiable Rewards. *arXiv preprint arXiv:2602.01601*, 2026. URL https://arxiv.org/abs/2602.01601.

Pixel Nomand, Elena Voss, Marcus Hale, and Sofia Reyes. PAIR: Pairwise-Aware Inclusion Reweighting for Adaptive Rollout Allocation in RLVR. *arXiv preprint arXiv:2608.11368*, 2026a. URL https://arxiv.org/abs/2608.11368.

Pixel Nomand, Elena Voss, Marcus Hale, and Sofia Reyes. Early Verdicts, Better Budgets: Sequential Adaptive Rollout Allocation for Compute-Efficient RLVR. *arXiv preprint arXiv:2607.26253*, 2026b. URL https://arxiv.org/abs/2607.26253.

Michael Noukhovitch, Hamish Ivison, Nathan Lambert, and Aaron Courville. Learning to Solve Hard Problems in RL for LLMs by Never Giving Up. *arXiv preprint arXiv:2609.13443*, 2026. URL https://arxiv.org/abs/2609.13443.

Kishan Panaganti, Zhenwen Liang, Wenhao Yu, Haitao Mi, and Dong Yu. Group Distributionally Robust Optimization-Driven Reinforcement Learning for LLM Reasoning. *arXiv preprint arXiv:2601.19280*, 2026. URL https://arxiv.org/abs/2601.19280.

Chansung Park, Juyong Jiang, Fan Wang, Sayak Paul, Jiasi Shen, Jing Tang, and Jianguo Li. TAROT: Test-driven and Capability-adaptive Curriculum Reinforcement Fine-tuning for Code Generation with Large Language Models. *arXiv preprint arXiv:2602.15449*, 2026. URL https://arxiv.org/abs/2602.15449.

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

Shivchander Sudalairaj, Kai Xu, Akash Srivastava, and Giorgio Giannone. sGPO: Trading Inference FLOPs for Training Efficiency in RLVR. *arXiv preprint arXiv:2606.08854*, 2026. URL https://arxiv.org/abs/2606.08854.

Yifan Sun, Jingyan Shen, Yibin Wang, Tianyu Chen, Zhendong Wang, Mingyuan Zhou, and Huan Zhang. Improving Data Efficiency for LLM Reinforcement Fine-tuning Through Difficulty-targeted Online Data Selection and Rollout Replay. *arXiv preprint arXiv:2506.05316*, 2025. URL https://arxiv.org/abs/2506.05316.

Tao Wang, Shuo Li, Yan Sun, Dongsheng Ding, and Edgar Dobriban. Where to Spend Rollouts: Hit-Utility Optimal Rollout Allocation for Group-Based RLVR. *arXiv preprint arXiv:2605.07114*, 2026a. URL https://arxiv.org/abs/2605.07114.

Weiqi Wang, Xin Liu, Binxuan Huang, Hejie Cui, Rongzhi Zhang, Changlong Yu, Shuowei Jin, Jingfeng Yang, Qingyu Yin, Zhengyang Wang, Zheng Li, Yifan Gao, Priyanka Nigam, Bing Yin, Lihong Li, and Yangqiu Song. HeaPA: Difficulty-Aware Heap Sampling and On-Policy Query Augmentation for LLM Reinforcement Learning. *arXiv preprint arXiv:2601.22448*, 2026b. URL https://arxiv.org/abs/2601.22448.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Yongxin Wang, Dongchun Xie, Hanhui Li, Yiwei Wang, Xiaodan Liang, and Jing Tang. Depth-Breadth Synergy in RLVR: Unlocking LLM Reasoning Gains with Adaptive Exploration. *arXiv preprint arXiv:2508.13755*, 2025. URL https://arxiv.org/abs/2508.13755.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, 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, 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. URL https://arxiv.org/abs/2503.14476.

Yuheng Zhang, Wenlin Yao, Changlong Yu, Yao Liu, Qingyu Yin, Bing Yin, Hyokun Yun, and Lihong Li. Improving Sampling Efficiency in RLVR through Adaptive Rollout and Response Reuse. *arXiv preprint arXiv:2509.25808*, 2025. URL https://arxiv.org/abs/2509.25808.

Zhi Zhang, Zhen Han, Costas Mavromatis, Qi Zhu, Yunyi Zhang, Sheng Guan, Dingmin Wang, Xiong Zhou, Shuai Wang, Soji Adeshina, Vassilis Ioannidis, and Huzefa Rangwala. Train Less, Learn More: Adaptive Efficient Rollout Optimization for Group-Based Reinforcement Learning. *arXiv preprint arXiv:2602.14338*, 2026. URL https://arxiv.org/abs/2602.14338.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025a. doi: 10.48550/arXiv.2507.18071. URL https://arxiv.org/abs/2507.18071.

Haizhong Zheng, Yang Zhou, Brian R. Bartoldson, Bhavya Kailkhura, Fan Lai, Jiawei Zhao, and Beidi Chen. Act Only When It Pays: Efficient Reinforcement Learning for LLM Reasoning via Selective Rollouts. *arXiv preprint arXiv:2506.02177*, 2025b. URL https://arxiv.org/abs/2506.02177.

Zhanke Zhou, Xiangyu Lu, Chentao Cao, Brando Miranda, Tongliang Liu, Bo Han, and Sanmi Koyejo. The Easy, the Hard, and the Learnable: Confidence and Difficulty-Adaptive Policy Optimization for LLM Reasoning. *arXiv preprint arXiv:2606.07950*, 2026. URL https://arxiv.org/abs/2606.07950.

Tianshu Zhu, Wenyu Zhang, Xiaoying Zuo, Lun Tian, Haotian Zhao, Yucheng Zeng, Jingnan Gu, Daxiang Dong, Jianmin Wu, Dawei Yin, and Dou Shen. Rollout Pass-Rate Control: Steering Binary-Reward RL Toward Its Most Informative Regime. *arXiv preprint arXiv:2605.05112*, 2026. URL https://arxiv.org/abs/2605.05112.

Heming Zou, Qi Wang, Yun Qu, Yuhang Jiang, Lizhou Cai, Yixiu Mao, Ru Peng, Xin Xu, Weijie Liu, Kai Yang, Saiyong Yang, and Xiangyang Ji. TRACE: A Unified Rollout Budget Allocation Framework for Efficient Agentic Reinforcement Learning. *arXiv preprint arXiv:2606.11119*, 2026. URL https://arxiv.org/abs/2606.11119.

A APPENDIX

A.1 EXPERIMENTAL CONFIGURATION

Table A1: Training and sampling configurations.

Model, data, framework	
Model	Qwen3.6-27B (hybrid attention)
Train / eval data	SWE-bench Verified: 449 train problems / 51-problem eval slice
Reward	verifiable SWE grader (binary: patch passes fail-to-pass tests)
Framework	slime (SGLang rollout + Megatron train), image `slimerl/slime:latest`
Software	CUDA 12.9.1, NCCL 2.28.9
Hardware / parallelism	
Nodes / GPUs	2 nodes $\times$ 8 H200 = 16 GPUs (colocated actor+learner)
Parallelism	TP=4, PP=2, CP=2
SGLang mem frac.	0.5; sandbox concurrency = 6
RL recipe (shared)	
Global batch (gbs)	4 query-groups
Max tokens / GPU	8192 (per-GPU log-prob/gradient batching budget, training side)
Rollout context / response len	32,000 max context; max response len 4096; temperature 1.0
Optimizer / LR	Adam, lr 1×10^{-6}, constant
Advantage / clip	GRPO; eps-clip 0.2; KL coef 0; entropy coef 0
Rollouts / eval	num-rollout depends on rollout time; eval every 5 rollout steps
Sampling	
ARCA (ours) — event-driven streaming allocation	
Allocation strategy	initial n_0=8; resample only all-wrong groups; max 3 shots; refill when live-queries $<$ burst; exit at M=4 mixed groups
Sampling schedules	$(8,4,4)$ (cum. 8→12→16), $(8,6,4)$ (8→14→18), $(8,8,8)$ (8→16→24)
Burst widths	$(8,4,4)$: b8×8 or b16×8; $(8,6,4)$: b16×8; $(8,8,8)$: b8×8; max in-flight 24
GRPO (baseline) — fixed-N oversampling	
Group / oversample	N=16 or N=8 samples/query, $b = 16$ or $b = 8$ queries
Update	num-steps-per-rollout 32

A.2 PROOFS FOR STOPPING AND RESAMPLING

We use q for a query instance, j for a rollout, k for a shot, and K for the maximum number of resampling stages. The target number of mixed groups is M; J counts groups actually obtained. The failure probability is always $1 - p_q$.

Proof of Lemma 1. Under Assumption 1, the only homogeneous outcomes of N fresh rollouts are all passes and all failures. Their disjoint probabilities are p^N and $(1-p)^N$, proving Eq. (3). For $N \geq 2$,

$$m'_N(p) \;=\; N\big[(1-p)^{N-1} - p^{N-1}\big]. \tag{8}$$

This is positive below $1/2$ and negative above it, so $p = 1/2$ maximizes the fresh-group mixed probability.

If the retained pool contains only failures, it remains homogeneous exactly when all n new rollouts fail. Conditional independence gives probability $(1-p)^n$. For a pool containing only successes, the corresponding probability is p^n. Taking complements proves Eq. (4). Subtracting the fresh-group probability gives

$$h_0(n,p) - m_n(p) \;=\; p^n, \qquad h_1(n,p) - m_n(p) \;=\; (1-p)^n. \tag{9}$$

Both comparisons concern the same query and success probability. A different fresh query may have a different p and need not satisfy this comparison. □

The increase from planning one more additional rollout is

$$\begin{aligned} h_0(n+1,p) - h_0(n,p) &= (1-p)^n p, \\ h_1(n+1,p) - h_1(n,p) &= p^n(1-p). \end{aligned} \tag{10}$$

The previous n samples must all miss the opposite outcome for the next one to increase the event probability. These increments decrease with n even though each new rollout still has success probability p.

Proof of Proposition 1. Fix any potential reward sequence of length $S_K = N$. If it is homogeneous, both rules return $Y = 0$ and the staged rule reaches the cap. Otherwise, there is a first shot boundary $S_{k^\star}$ whose prefix is mixed. The staged rule closes there, and every longer prefix is also mixed. Thus both rules return $Y = 1$ and $C_q^{\text{both}} = S_{k^\star} \le N$.

Apply this argument to each query in the same ordered stream. The two rules accept the same query instances. Their Mth accepted query therefore has the same index, and summing their costs up to that index proves the target-M statement. A query's rollout saving is strict when its first mixed shot precedes the cap. □

Acceptance and cost of the two rules. Write $a_{\mathbf{n}}^{\text{both}}(p)$ and $c_{\mathbf{n}}^{\text{both}}(p)$ for the expected mixed indicator and cost when both homogeneous outcomes are resampled. A resampling stage is reached exactly when all previous rewards agree. Therefore

$$\begin{aligned} a_{\mathbf{n}}^{\text{both}}(p) &= m_{S_K}(p), \\ c_{\mathbf{n}}^{\text{both}}(p) &= n_0 + \sum_{k=1}^{K} n_k \big[p^{S_{k-1}} + (1-p)^{S_{k-1}}\big]. \end{aligned} \tag{11}$$

Under drop-all-correct, the query fails to produce a mixed group in two disjoint cases: all n_0 initial rollouts succeed, or all S_K potential rollouts fail. Equivalently, sum the disjoint events of first becoming mixed at each shot:

$$\begin{aligned} a_{\mathbf{n}}(p) &= m_{n_0}(p) + \sum_{k=1}^{K} \big[(1-p)^{S_{k-1}} - (1-p)^{S_k}\big] \\ &= 1 - p^{n_0} - (1-p)^{S_K}. \end{aligned} \tag{12}$$

Initial sampling always runs, and resampling stage k runs only if all S_{k-1} earlier rewards are zero. Hence

$$c_{\mathbf{n}}(p) = n_0 + \sum_{k=1}^{K} n_k (1-p)^{S_{k-1}}. \tag{13}$$

Subtracting the two rules' expressions gives the cost saving and acceptance loss:

$$\begin{aligned} c_{\mathbf{n}}^{\text{both}}(p) - c_{\mathbf{n}}(p) &= \sum_{k=1}^{K} n_k p^{S_{k-1}}, \\ a_{\mathbf{n}}^{\text{both}}(p) - a_{\mathbf{n}}(p) &= p^{n_0} - p^{S_K}. \end{aligned} \tag{14}$$

The lost groups have an all-correct initial group followed by at least one potential failure. An all-correct initial group therefore does not establish that further sampling has zero value for forming a mixed group.

Table A2: Resolution probabilities under early all-correct drop for $(8, 6, 4)$, conditional on p and no global interruption.

Resolution	Cumulative rollouts	Probability
Mixed after Initial sampling	8	$1 - p^8 - (1-p)^8$
Mixed after Resampling 1	14	$(1-p)^8 - (1-p)^{14}$
Mixed after Resampling 2	18	$(1-p)^{14} - (1-p)^{18}$
All-correct, dropped after Initial sampling	8	p^8
All-wrong, dropped after Resampling 2	18	$(1-p)^{18}$

For the $(8, 6, 4)$ schedule in Table A2,

$$\begin{aligned} a_{(8,6,4)}(p) &= 1 - p^8 - (1-p)^{18}, \\ c_{(8,6,4)}(p) &= 8 + 6(1-p)^8 + 4(1-p)^{14}. \end{aligned} \tag{15}$$

For Q query instances, let J_N count mixed groups under fixed-N sampling. Linearity of expectation gives

$$\mathbb{E}[J_N \mid p_{1:Q}] = \sum_{q=1}^{Q} m_N(p_q). \tag{16}$$

No independence between queries is needed for this identity. Let $J_{\mathbf{n}}$ count mixed groups under drop-all-correct $(8, 6, 4)$. On those same Q queries, the difference from fixed-16 is

$$\begin{aligned} \mathbb{E}[J_{\mathbf{n}} - J_{16} \mid p_{1:Q}] &= \sum_{q=1}^{Q} \big[a_{(8,6,4)}(p_q) - m_{16}(p_q)\big] \\ &= \sum_{q=1}^{Q} \big[p_q^{16} - p_q^8 + (1-p_q)^{16} - (1-p_q)^{18}\big]. \end{aligned} \tag{17}$$

The first pair of terms is the loss from early all-correct drop; the second is the gain from two extra rollouts beyond the baseline cap. Their sum can have either sign. Equal query counts also need not consume equal rollout budgets.

A.3 Mixed Groups Under a Rollout Budget

Proof of Proposition 2. For $0 < p < 1$, define the initial-sampling rate and the conditional resampling rate after failures:

$$\eta_0(p) = \frac{m_{n_0}(p)}{n_0}, \qquad g_n(p) = \frac{1-(1-p)^n}{n}. \tag{18}$$

Since $g_n(p) = p\, n^{-1} \sum_{\ell=0}^{n-1} (1-p)^\ell$, it is the average of a non-increasing sequence and decreases with n. For $n_k \le n_0$,

$$g_{n_k}(p) \ge g_{n_0}(p) = \eta_0(p) + \frac{p^{n_0}}{n_0} > \eta_0(p). \tag{19}$$

Put $w_k = (1-p)^{S_{k-1}}$. The ratio for drop-all-correct is a weighted average of the initial-sampling and conditional resampling rates:

$$\frac{a_{\mathbf{n}}(p)}{c_{\mathbf{n}}(p)} = \frac{n_0\eta_0(p) + \sum_{k=1}^{K} n_k w_k g_{n_k}(p)}{n_0 + \sum_{k=1}^{K} n_k w_k} \ge \eta_0(p). \tag{20}$$

If a resampling stage exists, it has positive weight and the inequality is strict. When both homogeneous outcomes are resampled, add the analogous terms with weights $p^{S_{k-1}}$ and rates $g_{n_k}(1-p)$. Symmetry of m_{n_0} gives the same lower bound and strictness.

It remains to compare fixed group sizes. Their increments

$$\Delta_N(p) := m_{N+1}(p) - m_N(p) = p^N(1-p) + (1-p)^N p \tag{21}$$

decrease with N, and $\Delta_2 = m_2/2$. If $\Delta_N \le m_N/N$, then

$$\frac{m_{N+1}}{N+1} = \frac{N}{N+1}\frac{m_N}{N} + \frac{\Delta_N}{N+1} \le \frac{m_N}{N}. \tag{22}$$

This weighted average is also at least $\Delta_N \ge \Delta_{N+1}$, completing the induction. Hence $m_N(p)/N$ is non-increasing for integers $N \ge 2$. At $p = 0$ or $p = 1$, every mixed probability is zero, so the stated weak bounds hold directly. □

Expected cost for a target of M groups. Consider a reference collector that draws independent query instances with replacement and completes one before starting the next. Under the fixed policy, let their success probabilities have distribution $\mathcal{D}$, and assume that the reward sequences of different sampled instances are independent. For drop-all-correct, define

$$\begin{aligned} \bar{a} &= \mathbb{E}_{P\sim\mathcal{D}}[a_{\mathbf{n}}(P)], \\ \bar{c} &= \mathbb{E}_{P\sim\mathcal{D}}[c_{\mathbf{n}}(P)], \\ \bar{m}_N &= \mathbb{E}_{P\sim\mathcal{D}}[m_N(P)]. \end{aligned} \tag{23}$$

Assume $\bar{a} > 0$ and $\bar{m}_N > 0$. Let F_M count the queries needed for M acceptances and let $C_M = \sum_{q=1}^{F_M} C_q$ be their total rollout cost. Then

$$\begin{aligned} \mathbb{E}[F_M] &= \frac{M}{\bar{a}}, \\ \mathbb{E}[C_M] &= M\frac{\bar{c}}{\bar{a}}, \\ \mathbb{E}[C_M^{\mathrm{GRPO}}] &= M\frac{N}{\bar{m}_N}. \end{aligned} \tag{24}$$

Proof. The indicators Y_q are independent Bernoulli variables with mean $\bar{a}$, so $\mathbb{E}[F_M] = M/\bar{a}$. Although a query's cost and acceptance can be dependent, the event $F_M \geq q$ depends only on earlier queries and is independent of the current cost C_q. Non-negativity then gives

$$\begin{aligned} \mathbb{E}[C_M] &= \sum_{q=1}^{\infty} \mathbb{E}\big[C_q\, \mathbb{I}\{F_M \geq q\}\big] \\ &= \bar{c}\sum_{q=1}^{\infty} \Pr(F_M \geq q) = \bar{c}\,\mathbb{E}[F_M]. \end{aligned} \tag{25}$$

For fixed-N sampling, each query costs N and is mixed with mean probability $\bar{m}_N$, giving the third identity. □

The expected ARCA cost is smaller exactly when

$$\frac{\bar{c}}{\bar{a}} < \frac{N}{\bar{m}_N} \quad\Longleftrightarrow\quad \frac{\bar{a}}{\bar{c}} > \frac{\bar{m}_N}{N}. \tag{26}$$

The same derivation applies to resampling of both outcomes after replacing $a_{\mathbf{n}}, c_{\mathbf{n}}$ by their corresponding functions. With equal caps, $\bar{a}^{\mathrm{both}} = \bar{m}_N$ and $\bar{c}^{\mathrm{both}} \leq N$, so Proposition 1 also gives a weak cost improvement for heterogeneous query populations.

For other caps or early all-correct drop, the distribution matters. For example, take $\Pr(P = 0) = 0.99$ and $\Pr(P = 1/2) = 0.01$. Drop-all-correct $(8, 6, 4)$ and fixed-16 give

$$\frac{\bar{a}}{\bar{c}} \approx 0.00055647 < 0.00062498 \approx \frac{\bar{m}_{16}}{16}. \tag{27}$$

The zero-success queries consume 18 rollouts under ARCA and 16 under GRPO. They are frequent enough to outweigh ARCA's higher rate on the $p = 1/2$ queries. This example shows why the fixed-p bound cannot be averaged into an unconditional benchmark guarantee. The production collector also launches overlapping bursts, which can add work outside the sequential reference cost C_M.

A.4 What Per-Query Dispatch Removes

Proposition 3 (Removing waits between shots)**.** *Fix a finite cohort, its reward outcomes, and the stopping rule. Start all initial shots at time zero. Let κ_q be query q's final shot and $d_{q,k} \geq 0$ the time to complete its shot k. Assume these durations do not change with dispatch order, every eligible shot has execution capacity, and no fresh queries are added. Put $\mathcal{I}_k = \{q : \kappa_q \geq k\}$. A synchronized*

sampler starts shot $k+1$ after every query in $\mathcal{I}_k$ completes shot k; it can accept mixed groups as soon as they return. For each mixed query, its readiness times satisfy

$$\begin{aligned} T_q^{\text{event}} &= \sum_{\ell=0}^{\kappa_q} d_{q,\ell}, \\ T_q^{\text{round}} &= \sum_{\ell=0}^{\kappa_q-1} \max_{i\in\mathcal{I}_\ell} d_{i,\ell} + d_{q,\kappa_q}, \\ T_q^{\text{event}} &\le T_q^{\text{round}}. \end{aligned} \tag{28}$$

If at least M queries become mixed, their Mth smallest readiness times satisfy $T_{(M)}^{\text{event}} \le T_{(M)}^{\text{round}}$.

Proof. Immediate dispatch starts the next shot when the previous one returns, giving the sum for T_q^{event}. A synchronized shot waits for the largest duration in $\mathcal{I}_k$. Thus

$$T_q^{\text{round}} - T_q^{\text{event}} = \sum_{\ell=0}^{\kappa_q-1} \left(\max_{i\in\mathcal{I}_\ell} d_{i,\ell} - d_{q,\ell} \right) \ge 0. \tag{29}$$

Each term is nonnegative because $q \in \mathcal{I}_\ell$ for its earlier shots. The inequality holds for every mixed query, hence also for the Mth smallest time over the same set. It is strict for a resampled query that finishes an earlier shot before that shot's slowest query. □

For example, an all-wrong initial group taking 1 time unit followed by a successful resampling stage taking 2 is ready at time 3. If another query's all-correct initial group is dropped after 10 time units, a shot barrier delays that resampling stage until time 10 and readiness until time 12. With target $M = 1$ and no other work, draining gives collection times 10 and 12.

The collector's end time is $T_{\text{collect}} = T_{(M)} + d_{\text{tail}}$, where d_{tail} is the remaining drain time after the target is reached. Earlier readiness improves collection time only when the readiness gain exceeds any increase in that tail. The proposition compares dispatch of the same staged work; fixed-N rollouts launched together and fresh-query top-ups require a separate timing comparison. Our experiments measure the resulting full-step time.

A.5 Per-Instance Evaluation and Cases

We fixed ARCA (8,6,4) $b = 16$, and GRPO $b = 16$, $N = 8$ checkpoints 99, 149, and 199 before inspecting their per-instance outcomes. The comparison matches rollout steps, not wall-clock time, tokens, or optimizer updates. ARCA uses the historical sampler that resamples both homogeneous outcomes. These cases therefore do not evaluate the later all-correct-drop rule.

The official dataset's four difficulty categories contain 22, 23, 5, and 1 of our 51 instances. We join labels by instance ID and verify repository and base commit. Each evaluation has one episode per instance at temperature zero. Saved episode segments repeat its reward; we merge them before counting and check the totals against the per-problem log. This avoids weighting an instance by its number of segments.

Table A3: Solved instances and paired outcomes at fixed checkpoints, out of 51.

Step	ARCA	GRPO	Both	ARCA only	GRPO only	Neither
99	30	2	2	28	0	21
149	27	12	12	15	0	24
199	23	15	14	9	1	27

At step 199, both methods fail to apply the patch for `psf__requests-1142`, and both have a runtime exception for `scikit-learn__scikit-learn-14710`. These count as not solved in Figure 6; removing the two shared failures leaves 23/49 versus 15/49 and the same nine-to-one split of exclusive successes. At step 149, ARCA additionally has a runtime exception on `sphinx-doc__sphinx-7590`. These are logged evaluation outcomes, not claims that every failure reflects an incorrect code edit.

We selected two of the six ARCA-only successes in the 15-minute–1-hour category for interpretable edits in different repositories, and include the sole GRPO-only success as a counterexample. No instance rated above one hour is solved by either method at these three checkpoints. The examples are therefore moderate-difficulty cases, not evidence of hardest-task gains. Code below abbreviates recorded tool edits; it is not a reconstructed final repository diff. Tool confirmations and evaluation rewards support the comparisons.

Task (paraphrased). Django 13590 (15 minutes–1 hour): repair `__range` lookups using a namedtuple without breaking ordinary tuples or lists.

```
GRPO: unpack every tuple/list
    return type(value)(*resolved)
    -> works for namedtuple; can break tuple/list

ARCA: unpack only namedtuples
    if hasattr(value, '_fields'):
        return type(value)(*resolved)
    return type(value)(resolved)

Recorded ARCA test cycle:
    initial unpacking -> 148 tests, 5 errors, 2 skipped
    conditional fix  -> 148 tests, OK, 2 skipped
Evaluation reward: ARCA 1; GRPO 0
```

Figure 9: Django namedtuple repair: ARCA revises an initial fix after a regression test exposes its effect on ordinary tuples. Recursive value resolution is omitted.

Both agents changed `Query.resolve_lookup_value`. ARCA first tried unpacking all values, then observed `TypeError: tuple() takes at most 1 argument (2 given)` in the expression tests. Its subsequent edit checks `_fields` before unpacking, preserving the iterable constructor used by ordinary tuples and lists. GRPO's recorded edit keeps unconditional unpacking,

and its submitted patch receives reward zero. This case shows a successful test-and-revise sequence in one ARCA evaluation, not a measured increase in that behavior's frequency.

Task (paraphrased). SymPy 13551 (15 minutes–1 hour): `Product(n + 1/2**k, (k, 0, n-1))` gives the wrong symbolic result. For $n = 2$, the product should be $(2 + 1)(2 + 1/2) = 15/2$, not $9/2$.

```
ARCA, in sympy/concrete/products.py:
    if q.is_Number:
        return None
    -> removes the invalid sum of separate products
    -> leaves the symbolic product unevaluated
    -> substitute n=2, then doit(): 15/2
    -> product tests: 18 passed

GRPO:
    head -100 /testbed/sympy/core/product.py
    -> No such file or directory
    -> repeats this lookup; submitted diff is empty
Evaluation reward: ARCA 1; GRPO 0
```

Figure 10: SymPy product repair: ARCA changes the evaluation rule and checks a concrete value; GRPO submits no patch after repeated unsuccessful file lookups.

The saved ARCA edit replaces a sum of per-addend products with `return None` in the unsupported branch. The product remains symbolic until concrete limits permit evaluation. The recorded product tests pass, but a broader sums/products test run reports 47 passes, one expected failure, and seven exceptions; we do not treat that broader run as clean. GRPO repeatedly looks for `sympy/core/product.py`, rather than `sympy/concrete/products.py`, and the evaluator records zero diff bytes.

Task (paraphrased). pytest 6202 (less than 15 minutes): preserve dots and brackets inside a parametrized test name in the failure report headline.

```
ARCA's recorded edit:
    return s.replace(".[", "[", 1)
    -> still alters the first matching parameter substring

GRPO's recorded edit:
    return s
    -> preserves the test name as constructed

For s = "test_boo[..[]":
    ARCA expression -> "test_boo[.[]"
    GRPO expression -> "test_boo[..[]"
Evaluation reward: ARCA 0; GRPO 1
```

Figure 11: Reverse case: GRPO removes the lossy replacement; limiting it to one occurrence in ARCA does not fix the reported input.

The per-instance CSV, extraction scripts, decoded-tool evidence, and source checksums are retained with the figure assets. These selected traces cannot establish that resampling caused a particular repair strategy, nor that ARCA reliably outperforms GRPO on any difficulty category across seeds.

A.6 Additional Training-Stability Diagnostics

Coverage and counting. Figure 12 supplies the full set of inspected GRPO composition traces behind Section 6: ten exhibit collapse or marked deterioration, and one shows no observed collapse within its logged horizon. Recovery can be temporary; these histories do not all follow a monotonic decline. Figures 13 and 14 report per-run loop diagnostics. Their GRPO coverage comprises ten runs, whereas the main pooled loop figure uses 13 log segments; these summaries should not be read as identical populations or as 13 independent training seeds.

For the main pooled comparison, each window contains two evaluations of 51 tasks per log: 1,326 episodes for GRPO and 918 for ARCA. GRPO has 31 flagged episodes at the start (10 solved) and 497 at the end (52 solved); ARCA has eight (one solved) and 45 (17 solved), respectively. A flagged episode counts once regardless of the number of repeated calls. Loop solve rate is conditional success in each window, not the probability of escaping a particular loop. Repetition need not prevent success: at step 34, the non-collapsing GRPO trace solves 18 of 29 flagged episodes. The plotted 95% Wilson intervals describe episode proportions, without accounting for dependence across repeated tasks or related logs.

A diagnose-without-editing failure. For `django__django-12143`, a failed evaluation at checkpoint 99 contains 21 recorded segments. The agent identifies that line 1634 of `options.py` requires `re.escape(prefix)` and announces an edit, but repeatedly calls `Bash` to re-read that line instead. The environment returns the unchanged code, prompting the same diagnosis and unexecuted intention until the episode exhausts its budget. The recorded response-token count grows from 21,156 to 31,980; segments 10 and 19 nevertheless contain byte-identical response text, separated by roughly 5,000 tokens of additional context. A second failed case, `astropy__astropy-14369`, repeats `Read`/`find` calls instead of executing its announced edit; this second case was not checked for byte-identical text. These examples illustrate one manifestation of collapse, not an exhaustive classification of failures.

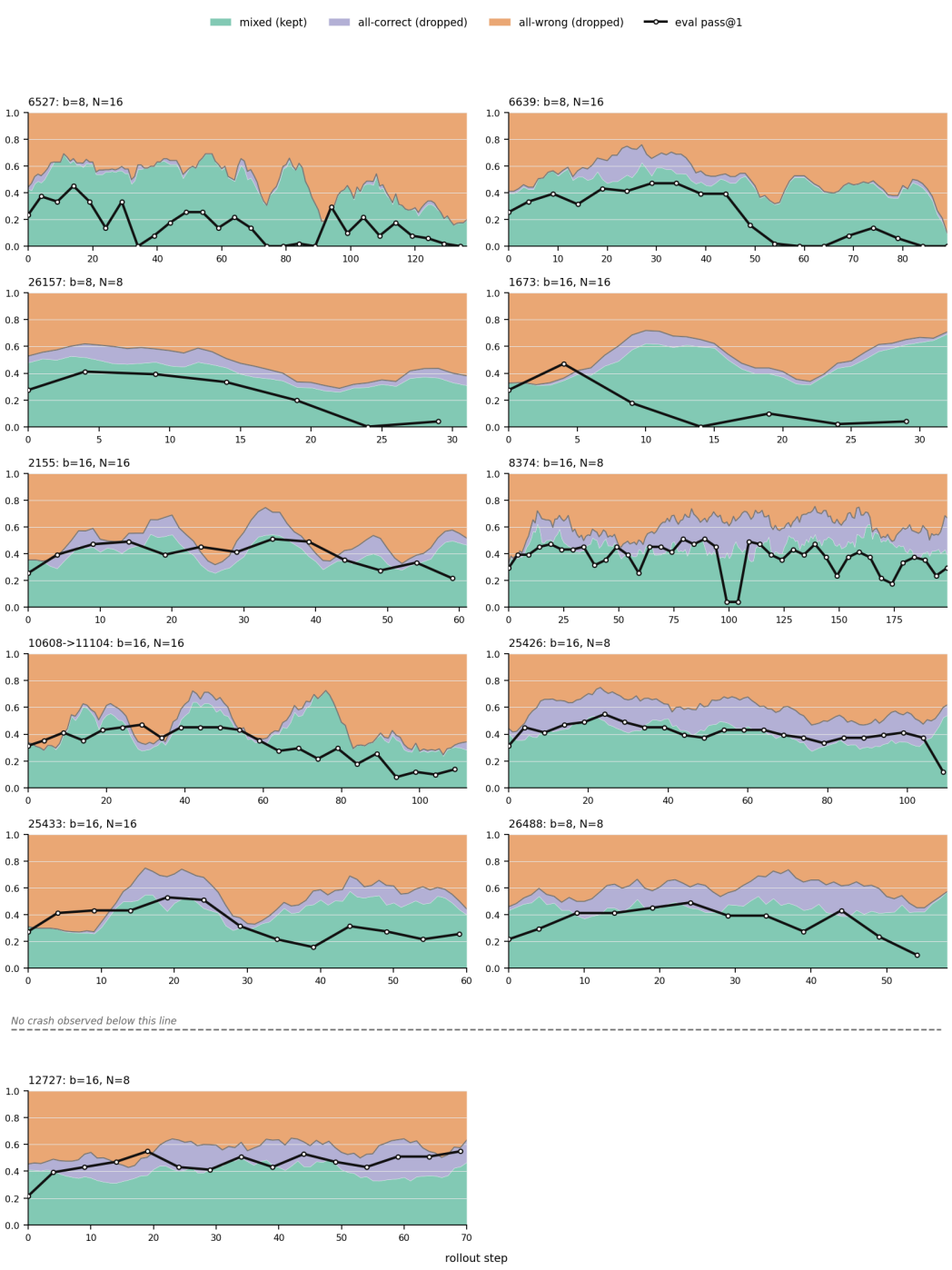


Figure 12: Full GRPO group-composition traces from the diagnostic analysis. The panels above the dashed divider show collapse or marked deterioration; the single panel below it shows no observed collapse over its logged horizon. Stacked areas distinguish mixed, all-correct, and all-wrong groups; the black curve is evaluation pass@1. Original diagnostic identifiers are retained in this appendix for traceability.

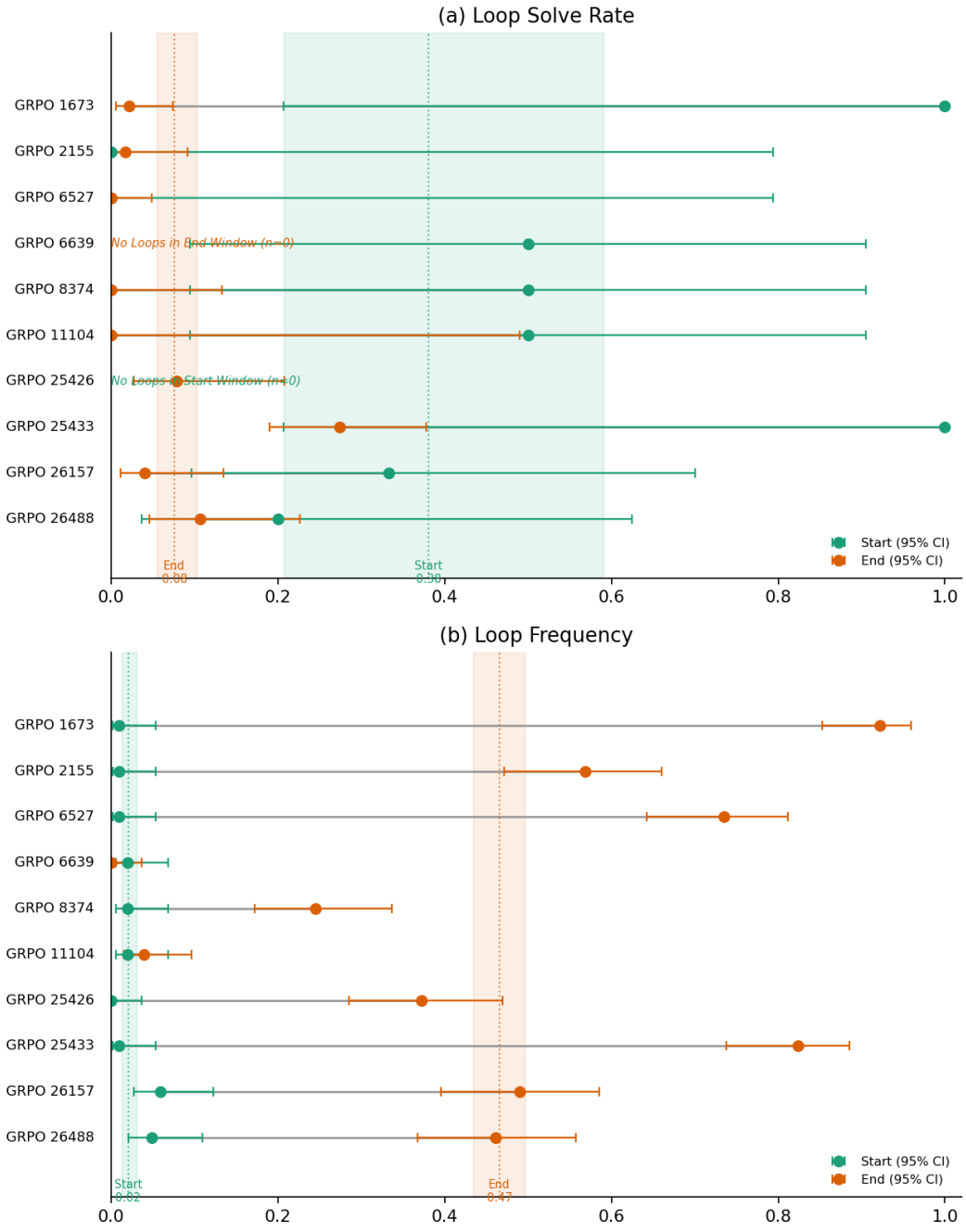


Figure 13: Per-run GRPO loop diagnostics for ten runs, comparing the first two and last two logged checkpoints. Bars show 95% Wilson intervals. Most runs show increasing loop frequency and decreasing conditional success, with exceptions and undefined solve rates when no episode is flagged. The aggregate guide lines summarize these ten run-level records, separately from the 13 log segments in Figure 8.

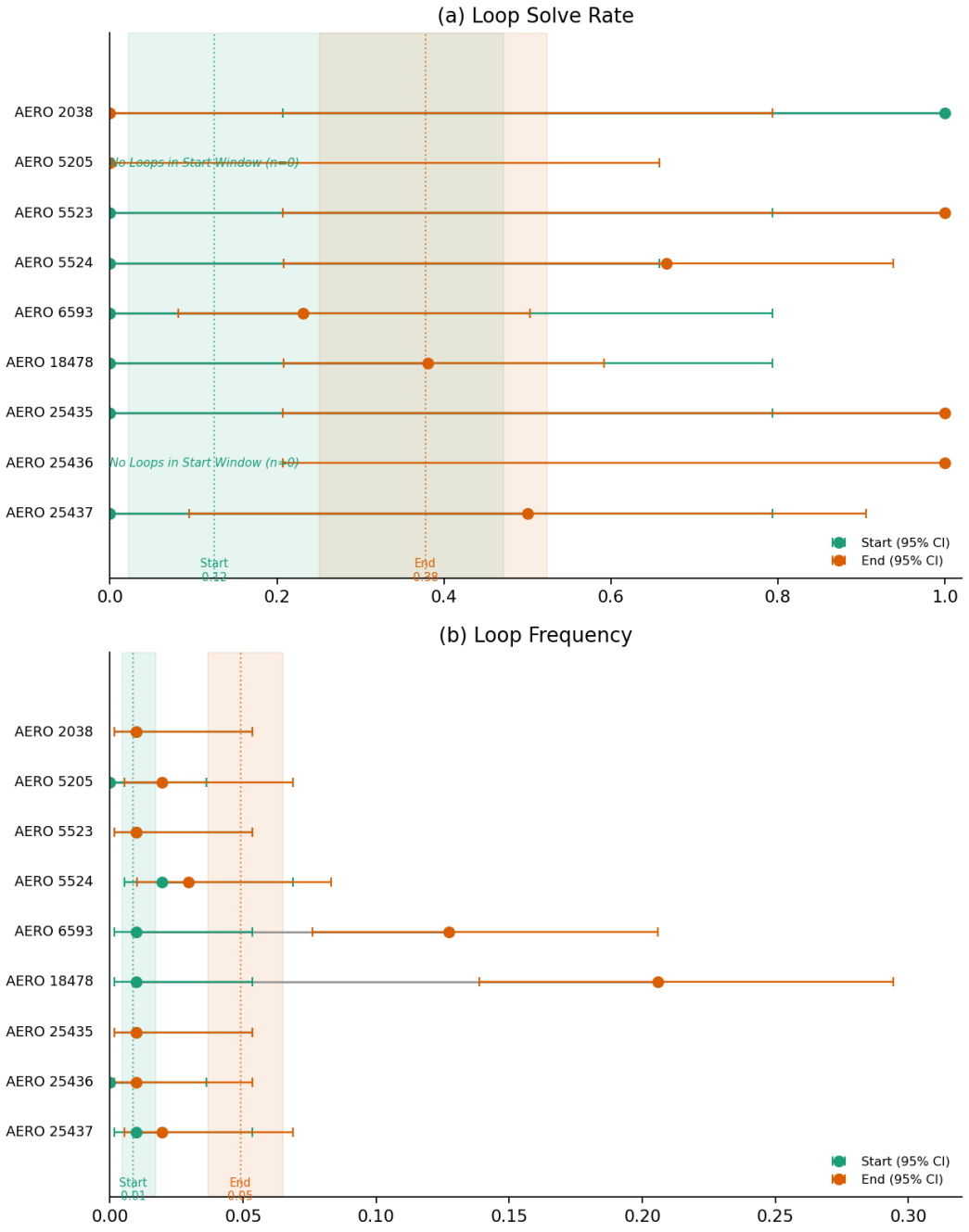


Figure 14: Per-run ARCA loop diagnostics for nine runs, with the same start/end windows and 95% Wilson intervals. Individual trajectories vary, and some loop frequencies increase, but the pooled change is substantially smaller than for GRPO. Conditional solve rates can be imprecise because few episodes are flagged.