CABRA: COST-AWARE BAYESIAN ROLLOUT ALLOCATION

Anonymous authors*
Paper under double-blind review

ABSTRACT

Group-relative reinforcement learning can spend much of its computation on response groups whose rewards barely differ. Discarding such groups after generation does not recover their sampling cost, and searching until a fixed number of useful groups is found can require many additional prompts. We introduce Cost-Aware Bayesian Rollout Allocation (CABRA), which fixes each prompt's number of responses before generation. From each prompt's past outcomes, it estimates, per expected token, the probability that a group will contain both correct and incorrect responses. A fixed number of top-ranked prompts receive complete groups of the full dense size. Skipped prompts with past outcomes then receive two-response refills if their expected token cost fits the remaining budget. All groups are generated once, without searching for replacements. On Qwen2.5-1.5B and GSM8K, CABRA uses 38.5% less adapter-backward work (LoRA backward-pass FLOPs) than dense DAPO across five controlled seeds, with 623.2 versus 621.4 mean correct answers out of 1,319. Without refills, work falls by 52.2% with 622.2 mean correct answers, so most of the saving comes from limiting the number of complete groups, and refills spend part of it. Generated tokens follow the same ordering. The accuracy differences are not statistically significant. We also compare CABRA with five external allocators in an eight-prompt protocol.

1 INTRODUCTION

A response group can be expensive to generate and still contain only correct or only incorrect answers. Reinforcement learning with verifiable rewards (RLVR) trains language models from automatically checked answers (Lambert et al., 2024). Group Relative Policy Optimization (GRPO) and Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO) standardize rewards within groups of responses to the same prompt (Shao et al., 2024; Yu et al., 2025). Under binary rewards, an all-correct or all-incorrect group has zero relative advantage, even though every response had to be generated.

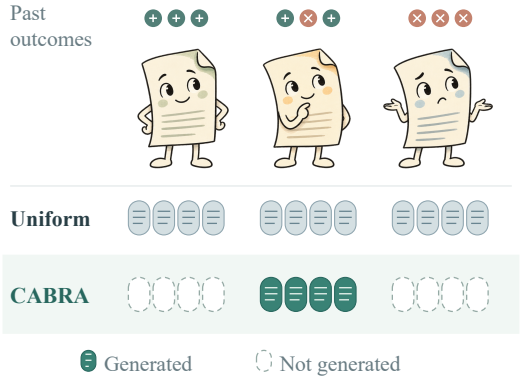

Figure 1: Illustration of history-guided rollout allocation. Filled slots denote generated responses; dashed slots are skipped.

Filtering such groups after generation does not recover that cost, and searching for replacement groups makes the workload depend on how many extra prompts must be tried. Filtering prompts by their history before generation can shorten this search (Zheng et al., 2025; Wu et al., 2026); other allocators use probe responses from the same step, predicted success rates, or budget optimization over the whole batch (Wang et al., 2026; Nguyen et al., 2026; Zhang et al., 2026). We instead fix every response count before generation.

*Code: `https://anonymous.4open.science/r/CABRA-D2EE/`

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.

We introduce Cost-Aware Bayesian Rollout Allocation (CABRA). Recurring prompts build up different histories, and not all of them need a complete group, i.e., the full number of responses that dense training samples per prompt (Figure 1). A Beta–Bernoulli model of each prompt's past outcomes estimates the probability that a new group will contain both correct and incorrect answers. CABRA ranks prompts by this probability divided by the expected token cost and gives complete groups to a fixed number of top-ranked prompts. The budget sets how many groups are generated, and history decides which prompts receive them. The selected groups are generated and used for training even if all their responses receive the same reward; no replacements are searched for.

In the same allocation, CABRA can also refill skipped prompts. A refill gives two responses to a skipped prompt with past outcomes if its expected token cost fits the remaining budget. Refills never remove a complete group; they spend part of the budget saved by generating fewer complete groups. The verifier, reward, objective, and optimizer stay the same within each comparison.

Across five controlled seeds on Qwen2.5-1.5B and GSM8K (Qwen et al., 2024; Cobbe et al., 2021), CABRA reduces mean adapter-backward work by 38.5% relative to dense DAPO, with 623.2 versus 621.4 mean correct answers out of 1,319. This metric estimates the training FLOPs spent on the responses used in the update. Without refills, the reduction grows to 52.2% with 622.2 mean correct answers, so most of the saving comes from selecting fewer complete groups. Four confirmation seeds show the same ordering for generated tokens, which fall by 37.9% for CABRA and by 51.8% without refills. Refills thus give responses to more prompts at a cost between no refills and dense DAPO. In an eight-prompt protocol, we compare CABRA with five external allocators to see whether the reduction holds when several prompts compete for the remaining budget.

Our contribution is to make two decisions separately, namely how many complete groups to generate and which prompts receive them. The budget left after these complete groups, measured in expected tokens, then pays for refills. Controlled ablations measure the work each decision adds or removes, and an eight-prompt comparison tests the full procedure against external allocators. Measurements of generated tokens, mixed groups (groups whose responses do not all receive the same reward), and refill timing show which groups reach training and at what cost.

2 CABRA: Budgeted Allocation Before Generation

Figure 2 shows CABRA's three steps: read correctness and length histories, select complete groups up to a response target (a fixed fraction of the dense response count), and refill skipped prompts that have past outcomes if their expected token cost fits the budget left after Step 2. Refills never change the complete groups. The allocated responses are generated and verified once and trained with the unchanged policy objective; their outcomes and lengths update only later histories.

2.1 RLVR training setup

Let i index prompts and j index responses. For prompt x_i, a dense RLVR trainer samples G responses $y_{i1}, \ldots, y_{iG}$ and assigns verifier rewards r_{ij}. Let g_i be the assigned group size, $\overline{r}_i$ its mean reward, $\mathrm{std}(\cdot)$ the within-group standard deviation, and $\epsilon > 0$ a small constant that prevents division by zero. The relative advantage of response j is

$$A_{ij} = \frac{r_{ij} - \overline{r}_i}{\mathrm{std}(r_{i1}, \ldots, r_{ig_i}) + \epsilon}. \tag{1}$$

We call the prompts drawn for a rollout its *primary prompts*. Extra prompts examined by search-based methods are *candidate prompts*. *Selected response groups* are the groups that enter the policy objective, and a group is *mixed* when at least two of its responses receive different policy rewards. Appendix A.1 gives each protocol's standard-deviation and ϵ conventions. CABRA changes only g_i. For each prompt, its history stores positive success and failure pseudocounts α_i, β_i (past correct and incorrect counts plus a prior), the number $N_i \geq 0$ of observed binary outcomes, and the expected response length $\ell_i \geq 1$ in tokens. The estimated success probability is $\widehat{p}_i = \alpha_i/(\alpha_i + \beta_i)$.

2.2 Step 1: read prompt histories

At rollout t, let $\mathcal{H}_{t-1}$ contain each prompt's Beta counts, observation count, and length estimate available before this rollout. For the current batch $\mathbf{x}_t = (x_1, \ldots, x_n)$ of n prompts, the allocator $\mathcal{A}$

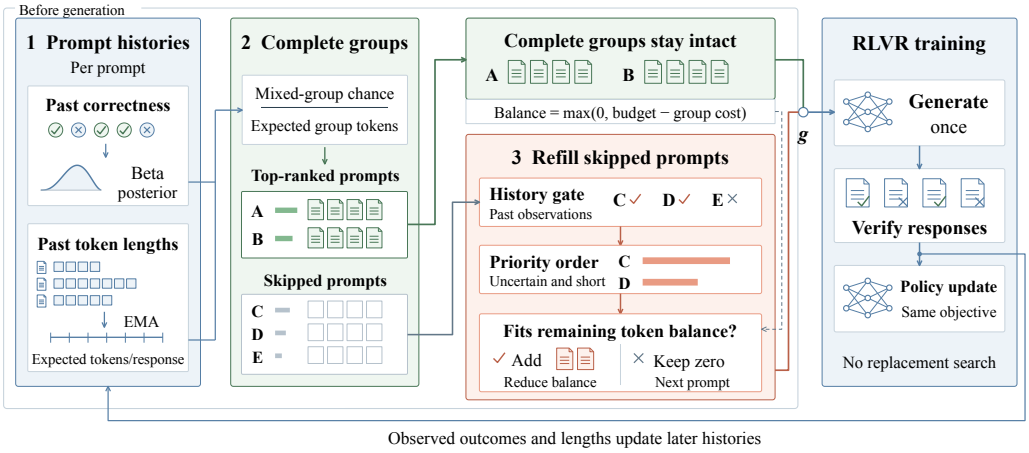

Figure 2: CABRA allocates complete groups (green) and refills (coral) before generation. Complete groups remain intact; skipped prompts with past outcomes are refilled in priority order if they fit the remaining token budget. Verified responses are used for training and update later histories. Document icons denote responses, empty slots denote no allocation, and all counts are schematic.

returns the group-size vector

$$\mathbf{g}_t = \mathcal{A}(\mathbf{x}_t, \mathcal{H}_{t-1}; G, \rho, \Delta, b). \tag{2}$$

Here ρ sets the response target for complete groups, Δ is the number of responses in one refill, and b sets the token budget for refills (defined below). After fixing $\mathbf{g}_t$, the trainer samples and verifies responses. A correct response increases α_i, an incorrect one increases β_i, either one increases N_i, and observed response lengths update an exponential moving average (EMA; Appendix A.1). These updates form $\mathcal{H}_t$, which is used on later visits and never for the current allocation.

The history records binary correctness even when the policy reward adds bonuses or penalties (reward shaping); the policy update uses the current rewards unchanged. A skipped prompt can be refilled only after at least one observed outcome, so the $\mathrm{Beta}(1, 1)$ prior alone cannot trigger a refill.

2.3 Step 2: Select Complete Groups

Let $p_i \sim \mathrm{Beta}(\alpha_i, \beta_i)$ be the posterior over the prompt's probability of a correct response. Conditional on p_i, suppose the G responses are independent Bernoulli trials. The posterior-predictive probability that the next complete group contains both correct and incorrect responses is

$$u_i(G) = 1 - \mathbb{E}[p_i^G] - \mathbb{E}[(1-p_i)^G], \qquad a_i = \frac{u_i(G)}{G\ell_i}. \tag{3}$$

The moments have the closed form $\mathbb{E}[p_i^G] = \prod_{k=0}^{G-1}(\alpha_i + k)/(\alpha_i + \beta_i + k)$; exchanging α_i and β_i gives the failure moment. The score therefore accounts for posterior uncertainty instead of using only the posterior mean, and its denominator is the expected token cost of the complete group. It is a cheap proxy for whether a group will mix correct and incorrect answers and does not predict gradient magnitude. As the policy changes, past outcomes can become outdated, so the posterior may be miscalibrated during training; we treat this as an empirical limitation.

Number and choice of complete groups. For a response fraction ρ, the rounded response target is $q_0 = \max\{1, \mathrm{round}(\rho n G)\}$. In all tested settings $q_0 \geq G$, and Step 2 assigns

$$k_0 = \min\{n, \lfloor q_0/G \rfloor\}, \qquad g_i^{(0)} = G\,\mathbf{1}\{i \text{ is among the top } k_0 \text{ scores } a_i\}. \tag{4}$$

Ties keep the original batch order because the implementation uses a stable sort. With $\rho = 0.70$ and $G = 4$, eight prompts give $q_0 = 22$, $k_0 = 5$, and 20 responses; two prompts give $q_0 = 6$, $k_0 = 1$, and four responses. The target limits the number of responses, and length affects only the ranking, so Step 2 does not exactly optimize a token budget. For smaller targets, the implementation falls back to giving at least one prompt some responses (Appendix A.1).

No replacement search. All selected groups are used for training even if their responses turn out to be all correct or all incorrect. GRESO filters prompts by history, generates responses, drops groups with zero reward variance, and keeps examining candidate prompts until it reaches a target or a limit (Zheng et al., 2025). CABRA instead keeps the primary prompts it selected. Given the history, Step 2 assigns exactly $k_0 G$ responses, and their actual lengths determine their token cost. Work is saved directly because fewer groups are generated and no replacements are searched for.

Exact prompt histories. Histories are keyed by the exact prompt text and are not shared between similar prompts. Unseen prompts start from $\mathrm{Beta}(1, 1)$ and length 128, so their initial scores can tie. The design targets training pools in which prompts recur and makes no prediction for unseen prompts.

2.4 Step 3: refill skipped prompts

Let $\mathbf{g} = (g_1, \ldots, g_n)$ collect the assigned response counts, and let $g_i^{(0)} \in \{0, G\}$ be the count assigned by Step 2. Define

$$C(\mathbf{g}) = \sum_i g_i \ell_i, \qquad C_{\mathrm{dense}} = \sum_i G \ell_i \tag{5}$$

as the expected token cost of an allocation and of the all-G dense allocation, respectively. A budget ratio $b \in [0, 1]$ sets an integer response count Q and a token budget B for the refill step:

$$Q = \lceil b\, nG \rceil, \qquad B = Q \frac{C_{\mathrm{dense}}}{nG}. \tag{6}$$

Here $\lceil \cdot \rceil$ rounds up, so b affects the allocation only through the integer Q. For a fixed batch and history, ratios that give the same Q also give the same B. The refill step counts expected tokens instead of responses, so a refill for a prompt with short responses may fit where one with long responses does not.

Step 3 preserves complete groups even when their expected cost exceeds B. Refills use only the remaining budget,

$$R_0 = \max\{0,\, B - C(\mathbf{g}^{(0)})\}. \tag{7}$$

If $R_0 = 0$, Step 3 returns $\mathbf{g}^{(0)}$ unchanged. Each accepted refill reduces the remaining budget by its expected cost, so $C(\mathbf{g}) \le \max\{C(\mathbf{g}^{(0)}), B\}$. The response target limits Step 2, while B limits only the expected tokens added in Step 3; neither caps the tokens actually generated. Appendix A.1 gives the rounding conventions and the eight-prompt example with 20 initial responses and $Q = 22$.

Step 3 then considers the prompts that Step 2 skipped ($g_i^{(0)} < G$). A skipped prompt is eligible for a refill only if it has at least one observed outcome from an earlier rollout; we call this requirement the *history gate*. Eligible prompts receive the score

$$s_i = \frac{G \widehat{p}_i (1 - \widehat{p}_i)}{\sqrt{\ell_i + 1}}, \tag{8}$$

and are visited from highest to lowest score. The remaining budget R starts at R_0. A refill of Δ responses is assigned only if $\Delta \ell_i \le R$, and its expected cost is then subtracted from R. Otherwise, that prompt's assignment stays unchanged and scanning continues. The score s_i favors uncertain outcomes and short responses, and the cost test in this single greedy pass determines which refills fit. Algorithm 1 summarizes the allocation and training cycle; Algorithm 2 gives the refill pass in detail.

The refill score is proportional to $u(2; p) = 2p(1 - p)$, the probability that two responses include one correct and one incorrect answer, evaluated at the point estimate instead of averaged over the posterior as in Step 2. Its derivation is in Appendix G.2. The score order matters only when at least two eligible skipped prompts compete and the remaining budget cannot pay for all their refills, which can happen only in the eight-prompt protocol. Otherwise, every order gives the same group sizes. In the two-prompt study, only one prompt is skipped per step, so ordering has no effect there.

State and allocation cost. Each prompt stores two Beta counts, an observation count, and one length estimate, so N observed prompts need $O(N)$ stored state. Allocation scores n prompts, sorts them, and makes one pass to add refills. This takes $O(n \log n)$ time and needs no auxiliary predictor or extra model pass. Appendix E gives the state and cost accounting.

Algorithm 1 CABRA: one pre-generation allocation

Require: Primary prompts $\mathbf{x}$, history $\mathcal{H}$, full size G, response fraction ρ, refill size Δ, ratio b

1: Read $(\alpha_i, \beta_i, N_i, \ell_i)$ from $\mathcal{H}$ ▷ Step 1
2: Compute a_i and k_0 by Eqs. 3–4 ▷ Step 2
3: Give G responses to the top k_0 prompts; give zero to the others
4: $\mathbf{g} \leftarrow \mathbf{g}^{(0)}$
5: Add refills to eligible skipped prompts (Section 2.4) ▷ Step 3
6: Generate the assigned groups once; compute the fixed policy objective
7: Update prompt histories from the observed responses for later rollouts

3 Experimental Setup

We use two-prompt batches to study each allocation decision separately and eight-prompt batches to compare the full method with external allocators. With two primary prompts, Step 2 skips only one prompt, so the ablations can test the number of complete groups, the refill, and the history gate without skipped prompts competing for refills. With eight primary prompts, several skipped prompts can compete for the remaining budget. We compare external allocators in this setting and report its costs per rollout, separately from the per-step costs of the two-prompt study.

Model and evaluation. We fine-tune the 1.5-billion-parameter Qwen2.5-1.5B-Instruct (Qwen et al., 2024) with rank-8 Quantized Low-Rank Adaptation (QLoRA) (Hu et al., 2022; Dettmers et al., 2023) on a 256-prompt Grade School Math 8K (GSM8K) training pool (Cobbe et al., 2021). Training uses temperature 1.0, a 320-token limit, and top-$p = 1.0$ for eight prompts or 0.95 for two prompts; Appendix A.4 defines the protocols. Evaluation uses greedy decoding and exact-match accuracy on all 1,319 GSM8K test questions. The model, reward, optimizer, and schedule are fixed within each comparison; quantization, adapter, optimizer, and reward constants are in Appendix A.1.

Two-prompt mechanism study. Each run trains for 300 steps with two prompts per step and dense group size four; each primary prompt is drawn independently from the pool. DAPO trains on two complete four-response groups. GRESO uses its history filter and candidate search to select two complete mixed groups. CABRA w/o refill omits Step 3; it ranks the two prompts by the Step 2 score and gives one of them a complete group. VIP redistributes the same eight-response budget using posterior success variance (Yu et al., 2025; Nguyen et al., 2026). CABRA without history uses the same refill size $\Delta = 2$ and budget $b = 0.68$ as CABRA, but has no history gate, so any skipped prompt can be refilled. Table 29 reports every per-seed correct-answer count; Appendix C reports uncertainty and additional allocation statistics. Seed 42 was used to choose the settings; seeds 43–46 are confirmation runs. The budget and refill-size controls use only seed 42 and are reported separately.

Eight-prompt comparison. We compare CABRA with dense DAPO, GRESO, Hit-Utility Optimal Rollout Allocation (HORA), Adaptive Efficient Rollout Optimization (AERO), and VIP, which allocates responses using predicted success rates (Yu et al., 2025; Zheng et al., 2025; Wang et al., 2026; Zhang et al., 2026; Nguyen et al., 2026). DAPO draws candidate prompts until it has selected eight mixed groups. HORA begins with two responses per prompt and assigns 16 additional responses by marginal hit utility. AERO adapts the number of responses per prompt and then curates the generated responses. VIP chooses group sizes from predicted success probabilities under a fixed response budget. CABRA uses a 70% response target for complete groups and gives two-response refills to eligible prompts under the $b = 0.68$ token budget. Each run has 75 rollouts and two disjoint optimizer minibatches per rollout, for 150 updates. All methods use the same model and initialization, primary-prompt sequence, DAPO objective, optimizer schedule, verifier, reward, hardware, horizon, and 1,319-question evaluation. Only the allocation policy changes, although search-based methods may examine additional candidate prompts. Generated and selected responses, mixed groups, and padded tokens are outcomes of each allocator and are not held equal across methods.

Cost and uncertainty. We measure two token counts. *Generated rollout tokens* count everything sampled, before any selection or curation. *Selected padded response tokens* count the response

positions, including padding, that enter the policy objective. Figure 4 reports the former. From the latter we compute *adapter-backward work*, the FLOPs of the backward pass through the Low-Rank Adaptation (LoRA) adapters, which is our main cost metric. Let r be the adapter rank, d the hidden width, L the number of layers, M the number of adapted attention projections, and T the number of selected padded response tokens. Adapter-backward work is

$$F_{\text{back}} = 6\, r\, d\, L\, M\, T, \tag{9}$$

with $r = 8$, $d = 1536$, $L = 28$, and $M = 4$. The factor 6 converts each selected token's rank–hidden products into FLOPs under this counting convention. Eq. 9 counts only work on selected padded response tokens and excludes generation, verification, optimizer overhead, and communication. For each run, we report final accuracy alongside its training work; Appendix B gives per-run measurements. Runs use one 40 GB NVIDIA A800 GPU.

Evaluation coverage. We include every finished run with predictions on the full test set under the stated configurations; incomplete runs are not imputed. The six-allocator comparison uses seed 46, chosen in advance and shared by all methods. Repeated comparisons use the four seeds shared by DAPO, CABRA, HORA, and VIP; AERO has two runs and GRESO one.

4 Results

We first separate refill costs from the savings of fewer complete groups, then compare external allocators and examine generated tokens, mixed groups, refill timing, and budget sensitivity.

4.1 Where the work reduction comes from

To find where CABRA saves work, we remove the refill step (Figure 3; Table 1). Across five paired seeds, CABRA w/o refill uses 8.29 GFLOPs of adapter-backward work per step versus 17.36 for DAPO, a 52.2% reduction, with 622.2 versus 621.4 mean correct answers. Selecting one complete group instead of two gives the largest reduction among the tested variants, and the cost ordering holds on every seed.

	42	43	44	45	46	Mean	SD
DAPO	18.12	17.12	17.20	17.01	17.36	17.36	0.44
GRESO	18.25	17.24	17.32	17.13	17.48	17.48	0.45
VIP	16.87	16.76	16.53	16.65	16.63	16.69	0.13
No history	11.74	12.11	11.76	11.80	11.96	11.87	0.16
CABRA	10.73	10.59	10.53	10.81	10.71	10.67	0.11
No refill	8.38	8.24	8.22	8.34	8.29	8.29	0.07

Figure 3: Adapter-backward GFLOPs per step on five seeds, with the mean and sample SD. Darker cells mean more work; the SD column is not shaded. The methods rank in the same order on every seed. "No refill" and "No history" are CABRA ablations.

Refills add 28.7% work over CABRA w/o refill but still leave a 38.5% saving over DAPO. Across paired seeds, CABRA averages 1.0 more correct answer than CABRA w/o refill (95% CI $[-6.02, 8.02]$, $p = 0.713$) and 1.8 more than DAPO (CI $[-0.59, 4.19]$). Both intervals include zero. We did not test equivalence or whether CABRA's ranking beats random or length-only selection at the same budget.

Paired accuracy–cost trade-offs. The four confirmation seeds also allow a joint comparison of accuracy and work (Table 25). Here, one run dominates another when it has at least as many correct answers and no more work, and is strictly better on at least one of the two. CABRA dominates DAPO on three of four seeds; on the fourth, it gives up one correct answer for lower work. Against VIP and against the no-history ablation, CABRA dominates on two seeds and gives up some accuracy for lower work on the other two. The no-refill ablation dominates CABRA twice; on the other two

Table 1: Two-prompt cost decomposition (300 steps). Correct answers and backward work use seeds 42–46; generated tokens and active groups use seeds 43–46. Entries are mean ± sample SD where shown. Work cut is relative to DAPO; active groups count prompts that receive responses.

Allocator	Correct ↑ / 1,319	Backward ↓ (GFLOPs/step)	Work cut (%)	Generated ↓ (1k tokens/run)	Active groups/step
DAPO	621.4 ± 2.30	17.36 ± 0.44	0.0	623.9 ± 5.3	2.000
VIP	624.6 ± 2.88	16.69 ± 0.13	3.9	604.5 ± 3.4	1.000
No refill	622.2 ± 4.21	$\mathbf{8.29} \pm 0.07$	**52.2**	$\mathbf{300.5} \pm 2.0$	1.000
No history	621.4 ± 5.98	11.87 ± 0.16	31.6	432.6 ± 5.8	1.823
CABRA	623.2 ± 1.92	10.67 ± 0.11	38.5	387.3 ± 4.5	1.543

seeds, it uses less work but answers fewer questions correctly. Complete groups without refills thus always use the least work here, while refills add work and improve accuracy only on some runs.

4.2 Comparison with external allocators

In the eight-prompt study, several skipped prompts can compete for the remaining refill budget. In the six-allocator run on the shared seed (Table 2), CABRA reaches 47.76% accuracy at 72.00 GFLOPs of adapter-backward work per rollout. The five external allocators use 95.17–104.76 GFLOPs with 47.31–48.75% accuracy. CABRA therefore uses 24.4–31.3% less work in this run, and its accuracy is within 0.99 percentage points of the highest.

Table 2: Six-allocator comparison on the common eight-prompt run (seed 46). All rows share the model, 75 rollouts, 150 updates, and full GSM8K evaluation.

Method	Correct / 1,319	Accuracy ↑ (%)	Backward ↓ (GFLOPs/rollout)
DAPO	638	48.37	103.95
GRESO	**643**	**48.75**	104.69
HORA	624	47.31	104.23
AERO	625	47.38	95.17
VIP	626	47.46	104.76
CABRA	630	47.76	**72.00**

Table 3 repeats the comparison over four seeds. CABRA reduces mean backward work by 16.1%, 22.8%, and 23.1% relative to DAPO, HORA, and VIP.

Table 3: Eight-prompt comparisons on paired seeds, mean ± sample SD. Each block uses only seeds run by all of its methods. Δ accuracy is CABRA minus the comparator in percentage points; work cut is relative to that comparator's mean. Work is in GFLOPs per rollout.

Comparator	Accuracy (%)	Backward work	Δ accuracy	Work cut (%)
Seeds 43–46*: CABRA* 47.69 ± 0.61% *accuracy,* 80.64 ± 15.90 *work*				
DAPO	48.09 ± 0.67	96.17 ± 15.86	-0.40	16.1
HORA	47.57 ± 0.40	104.46 ± 0.35	$+0.11$	22.8
VIP	47.55 ± 0.35	104.80 ± 0.36	$+0.13$	23.1
Seeds 44, 46*: CABRA* 48.14 ± 0.54% *accuracy,* 88.23 ± 22.96 *work*				
AERO	47.46 ± 0.11	95.62 ± 0.64	$+0.68$	7.7

CABRA uses less work than HORA and VIP on every paired seed, although the size of the saving varies: against HORA it ranges from 0.25% on seed 44 to roughly 30% on the other three. The four mean accuracies lie within 0.54 points of each other, and CABRA's mean is within 0.40 points of each of the others. On the two seeds with AERO runs, CABRA is 0.68 points more accurate and uses 7.7% less mean work. Per-seed outcomes appear in Appendix B.

CABRA thus uses less mean backward work when several skipped prompts compete for refills. Attributing this difference to individual allocation decisions would require an eight-prompt no-refill ablation and ranking controls at a matched budget. We next return to the two-prompt runs to examine generation cost and when refills occur.

4.3 Generated tokens and mixed groups

Figure 4 extends the cost comparison to generation. Across four confirmation seeds, CABRA generates 37.9% fewer tokens than DAPO, 90.9% fewer than GRESO, and 10.5% fewer than CABRA without history. Without refills, tokens fall further to 300.5k per run (51.8% below DAPO), versus 387.3k for CABRA. Tokens and backward work rank the methods in the same order on every seed, so CABRA saves both generation and training work.

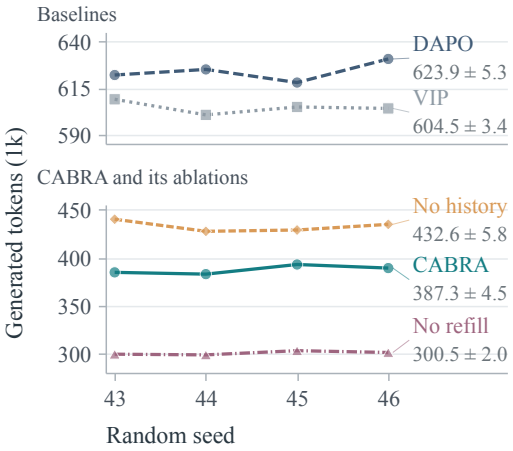

Figure 4: Generated tokens per run by seed; labels give mean and sample SD. The panels use different vertical scales. GRESO (not shown): 4.23–4.31M tokens/run.

CABRA also obtains more mixed groups per generated token, 1.165 per thousand generated tokens versus 0.961 for DAPO, a 21.2% increase computed from unrounded means (Table 30). Mixed groups measure the reward differences that training actually sees, whereas allocation uses only the binary correctness proxy. More mixed groups per token imply neither better test accuracy nor proportional wall-clock savings.

Similar mean accuracy does not mean that the models solve the same questions. Pooling the four paired runs, 72 question–run pairs are answered correctly only by CABRA and 64 only by DAPO (Table 18). The net difference is eight, and CABRA does not simply solve a subset of DAPO's questions. These pooled counts describe how individual predictions change; they are neither independent replications nor a significance test.

4.4 When refills happen

Figure 5 tests the history gate while keeping the refill size and budget fixed. Over the first 50 steps, CABRA refills on 14.4% of steps, versus 96.4% without history. The 82.0-point gap narrows to 4.4 and 4.8 points in the final two windows. Over all five runs, refills occur on 54.8% versus 82.1% of steps, and CABRA uses 10.1% less adapter-backward work than the version without history.

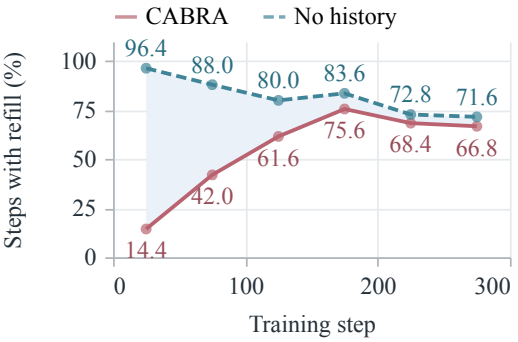

Figure 5: Share of steps with a refill per 50-step window, averaged over five runs; labels give window means. The gap is largest early in training.

This timing follows from how often prompts recur. Of the 600 primary prompts drawn in each two-prompt run, 61.3% repeat an earlier prompt on average (Table 24). A repeated prompt is eligible only if it received responses on an earlier visit. In the four confirmation runs with logged selections, the share of primary prompts with an observed outcome rises from 9.8% in the first 50-step window to 77.3% in the last. On average, half of the 256-prompt pool has an observed outcome by step 144 (Appendix E.6).

The history gate therefore acts as a warm-up period whose length depends on the data, unlike a fixed step schedule or a constant reduction in refill probability. Because the two-prompt setting has only one skipped prompt per step, these timing results do not depend on the refill score order.

4.5 REFILL BUDGET AND HISTORY

Table 4: Budget and history controls on seed 42 (300 steps). Q is the refill budget in responses, and b is the budget ratio. Δ correct and work cut are relative to DAPO on this seed, which gets 625 correct answers with 18.120 mean adapter-backward GFLOPs per step.

Q	History	b	Correct / 1319	Backward work (GFLOPs/step)	Δ correct	Work cut (%)
5	No history	0.55	624	8.376	-1	53.8
5	No history	0.60	621	8.376	-4	53.8
6	History-gated	0.68	626	10.728	$+1$	40.8
6	No history	0.68	627	11.738	$+2$	35.2
6	History-gated	0.75	625	10.888	0	39.9
6	No history	0.75	620	11.790	-5	34.9
7	No history	0.85	629	12.463	$+4$	31.2

With $nG = 8$, the seven settings in Table 4 give only three budgets, $Q \in \{5, 6, 7\}$. Budget ratios 0.55 and 0.60 both give $Q = 5$ and use 8.376 GFLOPs per step; $Q = 7$ raises work to 12.463 GFLOPs without history. At $Q = 6$, the history gate reduces work at both ratios, but it lowers accuracy at $b = 0.68$ and raises it at $b = 0.75$. The gate thus reduces spending without consistently improving accuracy. Runs with the same Q can still differ, because their sampled responses change later histories and length estimates. This control uses only one seed, so these differences may not hold on other seeds. Appendix D reports controls for the refill size and for the score order when only one prompt is skipped.

5 RELATED WORK

Selective rollout methods differ in when and what they allocate. GRESO filters prompts by history and then searches for mixed groups (Zheng et al., 2025); HIVE uses history and online verification to find useful prompts (Wu et al., 2026). CABRA instead fixes a limited number of complete groups, ranked by the posterior probability of a mixed group per expected token. HORA uses responses sampled in the same step, VIP predicts success rates to set group sizes across the batch, and AERO combines adaptive response counts with curation (Wang et al., 2026; Nguyen et al., 2026; Zhang et al., 2026). CERO, DynaMO, and Contextual Rollout Bandits explore cross-epoch allocation, advantage modulation, and learned scheduling (Zong et al., 2026; Fang et al., 2026; Lu et al., 2026). CABRA combines a limited number of complete groups with $0 \to 2$ refills for skipped prompts with history, all fixed before generation. Appendix G.6 extends the comparison.

6 LIMITATIONS

Evidence is limited to one model, GSM8K, and a recurring 256-prompt pool. The eight-prompt study lacks a no-refill ablation, and neither setting compares the ranking with random or length-only selection at a matched budget. Accuracy intervals include zero, and equivalence was not tested. Adapter-backward FLOPs exclude generation, verification, frozen-weight computation, and wall-clock overhead; generated tokens measure sampling. Exact-text histories do not transfer to unseen prompts, and policy changes can miscalibrate the posterior. Binary correctness is only a proxy. Shaped rewards can vary within all-correct or all-incorrect groups, so a group predicted to be all correct or all incorrect may still be useful. Further analyses appear in Appendices G.5 and F.

7 CONCLUSION

CABRA uses prompt histories to fix every response count before generation. Most of the saving comes from limiting the number of complete groups, and refills spend part of it without any replacement search. In the controlled study, CABRA reduces adapter-backward work by 38.5% and generated tokens by 37.9%, with small accuracy differences that are not statistically significant. It also uses less mean backward work than the external allocators. Choosing fewer complete groups before sampling, instead of replacing groups after seeing their rewards, reduces both generation and update work, and refills then give responses to some of the skipped prompts.

AI USE STATEMENT

AI tools assisted with drafting and polishing the manuscript and with retrieving and discovering related literature. The authors reviewed and verified all resulting text and references and take responsibility for the final manuscript.

REPRODUCIBILITY STATEMENT

The allocation equations and Algorithm 1 define the main mechanism. Appendix A.1 specifies history, rounding, and reward conventions; Appendix A.4 gives hyperparameters. Appendices B–D report seed-level comparisons and diagnostics. The accompanying anonymous code contains the allocator, configurations, test-data identity, and reported-result summaries. Appendix E.6 gives training and evaluation details.

REFERENCES

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. doi: 10.48550/arxiv.2110.14168.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning Ding. Process Reinforcement through Implicit Rewards. *arXiv preprint arXiv:2502.01456*, 2025. doi: 10.48550/arxiv.2502.01456. URL https://arxiv.org/abs/2502.01456.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Neural Information Processing Systems*, 2023. doi: 10.48550/arxiv.2305.14314.

Yangyi Fang, Jiaye Lin, Xiaoliang Fu, Cong Qin, H. Shi, Chaowen Hu, Lu Pan, Ke Zeng, and Xunliang Cai. How to allocate, how to learn? dynamic rollout allocation and advantage modulation for policy optimization. *arXiv preprint arXiv:2602.19208*, 2026. doi: 10.48550/arxiv.2602.19208.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving. *arXiv preprint arXiv:2309.17452*, 2023. doi: 10.48550/arxiv.2309.17452. URL https://arxiv.org/abs/2309.17452.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. *arXiv preprint arXiv:2103.03874*, 2021. doi: 10.48550/arxiv.2103.03874. URL https://arxiv.org/abs/2103.03874.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jian Hu, Xibin Wu, Wei Shen, Jason Klein Liu, Zilin Zhu, Weixun Wang, Songlin Jiang, Haoran Wang, Hao Chen, Bin Chen, Weikai Fang, Xianyu, Yu Cao, Haotian Xu, and Yiming Liu. OpenRLHF: An Easy-to-use, Scalable and High-performance RLHF Framework. *arXiv preprint arXiv:2405.11143*, 2024. doi: 10.48550/arxiv.2405.11143. URL https://arxiv.org/abs/2405.11143.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and H. Shum. Open-Reasoner-Zero: An Open Source Approach to Scaling Up Reinforcement Learning on the Base Model. *arXiv preprint arXiv:2503.24290*, 2025. doi: 10.48550/arxiv.2503.24290. URL https://arxiv.org/abs/2503.24290.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree Search for LLM Agent Reinforcement Learning. *arXiv preprint arXiv:2509.21240*, 2025. doi: 10.48550/arxiv.2509.21240. URL https://arxiv.org/abs/2509.21240.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 22 November 2024. doi: 10.48550/arxiv.2411.15124. URL https://arxiv.org/abs/2411.15124.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, H. Michalewski, V. Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving Quantitative Reasoning Problems with Language Models. *arXiv preprint arXiv:2206.14858*, 2022. doi: 10.48550/arxiv.2206.14858. URL https://arxiv.org/abs/2206.14858.

Gengsheng Li, Tianyu Yang, Junfeng Fang, Mingyang Song, Mao Zheng, Haiyun Guo, Dan Zhang, Jinqiao Wang, and Tat-Seng Chua. Unifying Group-Relative and Self-Distillation Policy Optimization via Sample Routing. *arXiv preprint arXiv:2604.02288*, 2026. doi: 10.48550/arxiv.2604.02288. URL https://arxiv.org/abs/2604.02288.

Jiazheng Li, Yawei Wang, David Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. SALT: Step-level Advantage Assignment for Long-horizon Agents via Trajectory Graph. *arXiv preprint arXiv:2510.20022*, 2025. doi: 10.48550/arxiv.2510.20022. URL https://arxiv.org/abs/2510.20022.

H. Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, I. Sutskever, and K. Cobbe. Let's Verify Step by Step. *arXiv preprint arXiv:2305.20050*, 2023. doi: 10.48550/arxiv.2305.20050. URL https://arxiv.org/abs/2305.20050.

Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, and Weizhu Chen. Rho-1: Not All Tokens Are What You Need. *arXiv preprint arXiv:2404.07965*, 2024. doi: 10.48550/arxiv.2404.07965. URL https://arxiv.org/abs/2404.07965.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-Like Training: A Critical Perspective. *arXiv preprint arXiv:2503.20783*, 2025. doi: 10.48550/arxiv.2503.20783. URL https://arxiv.org/abs/2503.20783.

Xiaodong Lu, Xiaohan Wang, Jiajun Chai, Guojun Yin, Wei Lin, Zhijun Chen, Yu Luo, Fuzhen Zhuang, Yikun Ban, and Deqing Wang. Contextual Rollout Bandits for Reinforcement Learning with Verifiable Rewards. *arXiv preprint arXiv:2602.08499*, 2026. doi: 10.48550/arxiv.2602.08499. URL https://arxiv.org/abs/2602.08499.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jian-Guang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. WizardMath: Empowering Mathematical Reasoning for Large Language Models via Reinforced Evol-Instruct. *arXiv preprint arXiv:2308.09583*, 2023. doi: 10.48550/arxiv.2308.09583. URL https://arxiv.org/abs/2308.09583.

Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. Orca-Math: Unlocking the potential of SLMs in Grade School Math. *arXiv preprint arXiv:2402.14830*, 2024. doi: 10.48550/arxiv.2402.14830. URL https://arxiv.org/abs/2402.14830.

Hieu Trung Nguyen, Bao Nguyen, Wenao Ma, Yuzhi Zhao, Ruifeng She, and V. Nguyen. Adaptive rollout allocation for online reinforcement learning with verifiable rewards. In *The Fourteenth International Conference on Learning Representations*, 2026. doi: 10.48550/arxiv.2602.01601. URL https://openreview.net/forum?id=Z5sWYACAop.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, S. Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, M. Simens, Amanda Askell, Peter Welinder, P. Christiano, Jan Leike, and Ryan J. Lowe. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 2022. doi: 10.52202/068431-2011.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. doi: 10.48550/arxiv.2412.15115.

Rafael Rafailov, Archit Sharma, E. Mitchell, S. Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Neural Information Processing Systems*, 2023. doi: 10.52202/075280-2338.

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. doi: 10.48550/arxiv.2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A Flexible and Efficient RLHF Framework. *arXiv preprint arXiv:2409.19256*, 2024. doi: 10.1145/3689031.3696075. URL https://arxiv.org/abs/2409.19256.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. OpenMathInstruct-1: A 1.8 Million Math Instruction Tuning Dataset. *arXiv preprint arXiv:2402.10176*, 2024. doi: 10.48550/arxiv.2402.10176. URL https://arxiv.org/abs/2402.10176.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022. doi: 10.48550/arxiv.2211.14275. URL https://arxiv.org/abs/2211.14275.

Peiyi Wang, Lei Li, Zhihong Shao, R. Xu, Damai Dai, Yifei Li, Deli Chen, Y.Wu, and Zhifang Sui. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. *arXiv preprint arXiv:2312.08935*, 2023. doi: 10.48550/arxiv.2312.08935. URL https://arxiv.org/abs/2312.08935.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, K. Dang, Xiong-hui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*, 2025. doi: 10.48550/arxiv.2506.01939.

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*, 2026. doi: 10.48550/arxiv.2605.07114. URL https://arxiv.org/abs/2605.07114.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *arXiv preprint arXiv:2203.11171*, 2022. doi: 10.48550/arxiv.2203.11171. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, F. Xia, Quoc Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv preprint arXiv:2201.11903*, 2022. doi: 10.52202/068431-1800. URL https://arxiv.org/abs/2201.11903.

Jiahao Wu, Ning Lu, Shengcai Liu, Kun Wang, Yanting Yang, Bailong Lin, Chen Jason Zhang, Li Qing, and Ke Tang. Train at moving edge: Online-verified prompt selection for efficient rl training of large reasoning model. *arXiv preprint arXiv:2603.25184*, 2026. doi: 10.48550/arxiv.2603.25184. URL https://arxiv.org/abs/2603.25184.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting Influential Data for Targeted Instruction Tuning. *arXiv preprint arXiv:2402.04333*, 2024. doi: 10.48550/arxiv.2402.04333. URL https://arxiv.org/abs/2402.04333.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. DoReMi: Optimizing Data Mixtures Speeds Up Language Model Pretraining. *arXiv preprint arXiv:2305.10429*, 2023. doi: 10.48550/arxiv.2305.10429. URL https://arxiv.org/abs/2305.10429.

Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. Logic-RL: Unleashing LLM Reasoning with Rule-Based Reinforcement Learning. *arXiv preprint arXiv:2502.14768*, 2025. doi: 10.48550/arxiv.2502.14768. URL https://arxiv.org/abs/2502.14768.

Shangyu Xing, Siyuan Wang, Chenyuan Yang, Xinyu Dai, and Xiang Ren. Lookahead Tree-Based Rollouts for Enhanced Trajectory-Level Exploration in Reinforcement Learning with Verifiable Rewards. *arXiv preprint arXiv:2510.24302*, 2025. doi: 10.48550/arxiv.2510.24302. URL https://arxiv.org/abs/2510.24302.

Kai Yan, Alexander G. Schwing, and Yu-Xiong Wang. Boosting Reinforcement Learning with Verifiable Rewards via Randomly Selected Few-Shot Guidance. *arXiv preprint arXiv:2605.15012*, 2026. doi: 10.48550/arxiv.2605.15012. URL https://arxiv.org/abs/2605.15012.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-Math Technical Report: Toward Mathematical Expert Model via Self-Improvement. *arXiv preprint arXiv:2409.12122*, 2024. doi: 10.48550/arxiv.2409.12122. URL https://arxiv.org/abs/2409.12122.

Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, Yudong Wang, Zijian Wu, Shuaibin Li, Fengzhe Zhou, Hongwei Liu, Songyang Zhang, Wenwei Zhang, Hang Yan, Xipeng Qiu, Jiayu Wang, Kai Chen, and Dahua Lin. InternLM-Math: Open Math Large Language Models Toward Verifiable Reasoning. *arXiv preprint arXiv:2402.06332*, 2024. doi: 10.48550/arxiv.2402.06332. URL https://arxiv.org/abs/2402.06332.

L. Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zheng Li, Adrian Weller, and Weiyang Liu. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. *arXiv preprint arXiv:2309.12284*, 2023. doi: 10.48550/arxiv.2309.12284. URL https://arxiv.org/abs/2309.12284.

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, Honglin Yu, Weinan Dai, Yuxuan Song, Xiang Wei, Haodong Zhou, Jingjing Liu, Wei Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yong-Xu Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Neural Information Processing Systems*, pp. 113222–113244, 2025. doi: 10.48550/arxiv.2503.14476. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping Reasoning With Reasoning. *arXiv preprint arXiv:2203.14465*, 2022. doi: 10.48550/arxiv.2203.14465. URL https://arxiv.org/abs/2203.14465.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. SimpleRL-Zoo: Investigating and Taming Zero Reinforcement Learning for Open Base Models in the Wild.

arXiv preprint arXiv:2503.18892, 2025. doi: 10.48550/arxiv.2503.18892. URL https://arxiv.org/abs/2503.18892.

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. doi: 10.48550/arxiv.2602.14338. URL https://arxiv.org/abs/2602.14338.

Haizhong Zheng, Yang Zhou, Brian R. Bartoldson, B. Kailkhura, Fan Lai, Jiawei Zhao, and Beidi Chen. Act only when it pays: Efficient reinforcement learning for llm reasoning via selective rollouts. In *Neural Information Processing Systems*, pp. 124321–124346, 2025. doi: 10.48550/arxiv.2506.02177. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/b42fb82fc3ffb88b872c4714f093875d-Abstract-Conference.html.

Yiming Zong, Yige Wang, and Jiashuo Jiang. Cross-epoch adaptive rollout optimization for rl post-training. *arXiv preprint arXiv:2606.05606*, 2026. doi: 10.48550/arxiv.2606.05606.

A Method Definition and Structural Properties

A.1 Implementation Details

Let i index a prompt in a batch of n prompts, G be the full response-group size, $g_i^{(0)} \in \{0, G\}$ be its initial complete-group assignment, and g_i be its final response count. We call $\mathbf{g}^{(0)}$ the *base allocation*: the Step 2 allocation, in which each prompt receives either all G responses or none. Refills add responses to it. For that prompt, CABRA stores success and failure pseudocounts α_i and β_i, an observation count N_i, and an exponential moving average (EMA) ℓ_i of earlier response lengths, with $\ell_i \geq 1$ because every generated response contains at least one token. The counts start from a uniform Beta$(1, 1)$ prior, and the Beta–Bernoulli estimate $\widehat{p}_i = \alpha_i/(\alpha_i + \beta_i)$ predicts response success. A budget ratio b gives the response count $Q = \lceil bnG \rceil$, rounded up, and the refill token budget $B = QC_{\text{dense}}/(nG)$, where $C_{\text{dense}} = \sum_i G\ell_i$ is the expected cost of assigning all G responses. The refill increment is fixed at $\Delta = 2$; the applied increment for an under-filled prompt is $\delta_i = \min\{\Delta, G - g_i^{(0)}\}$. The full-group contrast estimate $u_i(G)$ (the probability of both a success and a failure) and the priority score $s_i = G\widehat{p}_i(1 - \widehat{p}_i)/\sqrt{\ell_i + 1}$ determine which skipped prompts are refilled. A prompt becomes *eligible* for refill (it passes the history gate) after at least one prior binary outcome. The base allocation has its own response target. Refill computes the remaining budget $R_0 = \max\{0, B - C(\mathbf{g}^{(0)})\}$ from the same length estimates, made before generation. If the base costs less than B, a positive budget remains for refills; if it costs exactly B, no refill is added; and if it costs more than B, it is kept unchanged and nothing is added. We therefore do not assume $C(\mathbf{g}^{(0)}) \leq B$; B limits new refill spending and never reduces a base assignment.

Prompt and group terminology. A *primary prompt* occupies one of the n positions supplied at the start of a rollout. A *candidate prompt* is an additional prompt examined by a search-based allocator. A *selected response group* is a generated group included in the policy objective, and *selected padded response tokens* are the padded response positions in those groups. A *complete* (full) group has all G responses, and a *partial* group has fewer. A *refill*, also called an increment, adds δ_i responses to a prompt that the base allocation left under-filled. The *residual* is the part of the refill budget B that has not yet been spent. A group is *mixed* if its responses do not all receive the same policy reward and *collapsed* otherwise; *reward contrast* means such reward differences within a group.

Discrete settings. Both protocols use full-group size $G = 4$ and refill increment $\Delta = 2$. The base first rounds its $0.70nG$ response target to the nearest integer and then fits only complete groups. For eight primary prompts, the nominal target is 22 and five groups fit, giving 20 selected responses. For two primary prompts, the nominal target is 6 and one group fits, giving 4 selected responses. This base target is separate from the upward-rounded refill reference: $Q = \lceil 0.68nG \rceil$ equals 22 in the eight-prompt protocol and 6 in the two-prompt protocol.

Minimum-active safeguard outside the tested regime. The implementation requests at least one active prompt. If complete-group packing produces none, it assigns the highest-ranked prompt $\min\{\max(G, 2), \max(2, q_0)\}$ responses, where q_0 is the rounded base target. Thus targets below G can produce a partial fallback group and a target below two can be exceeded to avoid a singleton. Neither tested protocol activates this safeguard: $q_0 \geq G$ and the packed base already contains an active group. The formulas and algorithms that give each prompt all or none of its responses describe only this tested regime.

Reward normalization and history reward. The relative-advantage equation in the main paper denotes the protocol-specific stabilized normalization. The eight-prompt protocol divides by the sample standard deviation with correction 1 plus $\epsilon = 10^{-6}$. The two-prompt protocol divides by the population standard deviation, with the denominator clamped below at 10^{-6}. Policy rewards also differ by protocol: the mechanism study uses shaped correctness, while the eight-prompt study uses $+1/-1$ final-answer correctness with a shared soft penalty near the response limit. In both cases the Beta history is updated from the binary correct/incorrect verifier outcome alone, without reward shaping.

Predicted and realized length. The allocation decision uses the length EMA before generating the current responses, and the new response lengths update the EMA for subsequent steps. After response j is generated with token length m_{ij}, the update is $\ell_i \leftarrow 0.9\ell_i + 0.1m_{ij}$ from an initial value of 128. When a group has several responses, this update is applied once per response in their fixed generation order. A batch can therefore use more or fewer tokens than predicted. Its realized lengths then update the estimate, allowing it to follow changes over training.

Fixed training settings. QLoRA uses 4-bit NormalFloat (NF4) quantization, rank 8, LoRA scale 16, and dropout 0.05 on the query, key, value, and output projections. We optimize with AdamW at learning rate 10^{-5}. DAPO standardizes rewards within each active group, clips the policy ratio asymmetrically at $(0.20, 0.28)$, and uses no Kullback–Leibler (KL) penalty in the eight-prompt study. Its policy reward is binary $+1/-1$ final-answer correctness plus one shared soft penalty near the response limit; the allocator's Beta history records correctness only, so length shaping cannot alter success counts. The eight-prompt protocol samples distinct primary prompts within each rollout and permits recurrence across rollouts. Seeds 43–46 provide repeated runs; the common comparison uses seed 46.

A.2 Structural Properties of Partial Refill

Below we show that the allocation rule keeps complete groups, changes its cost reference in discrete steps, and depends on candidate order only when increments compete for capacity.

Refill decomposition. Let $\mathbf{g}^{(0)} \in \{0, G\}^n$ denote the base allocation, which gives each prompt all G responses or none. For an under-filled prompt, define

$$\delta_i = \min\{\Delta, G - g_i^{(0)}\}. \tag{10}$$

A cost-feasible refill pass produces

$$g_i = g_i^{(0)} + z_i\delta_i, \qquad z_i \in \{0, 1\}, \tag{11}$$

where z_i can be one only if prompt i has previous outcomes and the increment fits the remaining refill-stage expected cost. This form separates the base decision from the refill decision.

Base preservation. Equation 11 implies $g_i \geq g_i^{(0)}$ for every prompt. If the base assigns a full group, then $g_i^{(0)} = G$, $\delta_i = 0$, and the final allocation remains $g_i = G$. CABRA therefore neither removes a group selected by the base nor transfers its responses to another prompt. It modifies only prompts that the base leaves under-filled. With $C(\mathbf{g}) = \sum_i g_i\ell_i$ denoting expected token cost, the increase is

$$C(\mathbf{g}) - C(\mathbf{g}^{(0)}) = \sum_i z_i\delta_i\ell_i \geq 0. \tag{12}$$

Each accepted refill adds its predicted cost without displacing a full group. This property holds regardless of how those groups were chosen, so any selector that gives each prompt all or none of its responses can pass $\mathbf{g}^{(0)}$ to the same refill stage. Defining

$$R_0 = \max\{0, B - C(\mathbf{g}^{(0)})\} \tag{13}$$

gives the budget available for refills. If the base already reaches or exceeds B, then $R_0 = 0$ and the output is exactly the base. Otherwise the cost of each accepted refill is subtracted from R_0, so every output obeys $C(\mathbf{g}) \leq \max\{C(\mathbf{g}^{(0)}), B\}$.

The budget changes in steps. Let $m = nG$ be the dense response count and $b \in [0, 1]$. For an integer $q \in \{1, \ldots, m\}$, the quota $Q(b) = \lceil bm \rceil$ is constant on intervals

$$\frac{q-1}{m} < b \leq \frac{q}{m} \quad \Longleftrightarrow \quad Q(b) = q. \tag{14}$$

At the endpoint $b = 0$, $Q(0) = 0$. All ratios in one such interval, which we call a *quota cell*, give the same Q, so the expected-cost target changes in steps. Let $\pi_1, \pi_2, \ldots, \pi_M$ denote the M eligible

candidates in score order, with ties broken by decreasing batch index as in the implementation. Sequential admission is described exactly by

$$z_{\pi_j} = \mathbf{1}\{\delta_{\pi_j}\ell_{\pi_j} \le R_{j-1}\}, \qquad R_j = R_{j-1} - z_{\pi_j}\delta_{\pi_j}\ell_{\pi_j}, \tag{15}$$

for $j = 1, \dots, M$, starting from Eq. 13. The accepted increment count is $K_{\text{cost}} = \sum_{j=1}^{M} z_{\pi_j}$. A refill that does not fit leaves the remaining budget unchanged, so a later, cheaper one can still be admitted. The response count therefore follows from predicted lengths and is not a second hard constraint.

The refill budget B changes in the same steps. Writing $\bar{\ell} = C_{\text{dense}}/(nG)$ gives

$$B = Q\bar{\ell}. \tag{16}$$

Given the same pre-rollout state, two values of b in the same quota cell therefore produce the same Q and B. Their decisions can differ only after their histories, length estimates, base allocations, or sampled trajectories have diverged. The settings that directly control refills are therefore Q, the refill size, and the eligibility rule.

Allocation uses only past observations. At step t, let $\mathcal{H}_{t-1}$ contain the Beta counts, response-length estimates, and observation counts accumulated before the current rollout. The allocation is a mapping

$$\mathbf{g}_t = \mathcal{A}(\mathbf{x}_t, \mathcal{H}_{t-1}, \mathcal{A}_0; G, \Delta, b), \tag{17}$$

where $\mathcal{A}$ is the allocator and $\mathcal{A}_0$ denotes its complete-group selection step. Using the ρ-controlled rule for $\mathcal{A}_0$ in Section 2.3 gives the single CABRA mapping in Eq. 2; b determines B through the integer Q, and $\mathbf{x}_t$ is the current prompt batch. Responses and verifier rewards from step t update $\mathcal{H}_t$ only after the allocation is chosen. They can affect later groups but never the group that produced them. Refill therefore precedes generation and leaves the verifier and group-relative objective unchanged.

History induces a warm-up. Let $N_i(t-1)$ be the number of outcomes previously observed for prompt i. The indicator

$$e_i(t) = \mathbf{1}\{N_i(t-1) > 0\} \tag{18}$$

indicates whether prompt key i has a generated outcome before step t. A prompt that was only visited and skipped still has $e_i(t) = 0$ and retains its base allocation. After its first observation, the key becomes eligible on subsequent visits. Because observation counts accumulate, eligibility for each key changes at most once, from zero to one. The set of eligible keys grows monotonically even though the candidate set in a batch does not: prompt sampling, the base allocation, and the remaining budget still determine which prompts are available for refill. Early in training the allocator therefore behaves almost like the base allocation alone, and partial groups become possible once the needed state has accumulated. The result is a warm-up whose length depends on the data, without a separate step schedule.

Minimal useful group size. For a singleton group, let r_{i1} be its verifier reward; the group mean then equals that reward. Under the population-variance or explicit zero-advantage convention, its stabilized relative advantage is

$$A_{i1} = \frac{r_{i1} - r_{i1}}{0 + \epsilon} = 0. \tag{19}$$

The sample standard deviation with correction one is undefined for a singleton; this identity does not apply that estimator to a size-one group. Both tested protocols avoid singleton allocations. A one-response refill cannot directly contribute a standardized group-relative update. With two responses, unequal rewards yield nonzero within-group deviations. Equal rewards still give zero signal, so a size-two group is the smallest group that *can* be useful; not every refill will be. The choice $\Delta = 2$ thus follows from the definition of the group-relative advantage and is not merely a systems convention.

Single- and multi-candidate regimes. When at most one under-filled prompt is eligible, the main paper's priority score cannot alter which prompt is selected. Refill reduces to history eligibility followed by expected-cost admission. This regime isolates the refill and its timing. With several

eligible skipped prompts, the same rule divides a shared budget: history decides which prompts may compete, the score orders them, and the remaining expected-cost budget limits how many partial groups are admitted. Score order matters only when at least two prompts compete and the remaining budget cannot cover them all; otherwise every order yields the same final group sizes.

If candidate costs are equal, the greedy pass selects the highest-scoring increments that fit. When expected lengths differ, the allocator follows its score order rather than solving a global knapsack problem, which jointly selects the highest-value subset under one cost boundary. This deterministic pass is the priority mechanism we test; it needs no auxiliary model and keeps the base allocation exactly. Preservation and admission do not depend on the priority formula, so a different score can change candidate order while the budget guarantee still holds.

A.3 Signal–Cost Interpretation

Partial refill weighs the probability of obtaining reward contrast against the cost of an additional group. The score estimates contrast from history and discounts it by expected response length.

Contrast probability. Suppose a binary verifier response is correct with probability p. A group of size $g \geq 1$ contains at least one outcome of each type with probability

$$u(g; p) = 1 - p^g - (1 - p)^g. \tag{20}$$

For an empty group define $u(0; p) = 0$ separately. This quantity is zero for $g = 1$, positive for $g \geq 2$ when $p \in (0, 1)$, and largest near the learning boundary $p = 1/2$. In a relative objective, all-equal outcomes give no binary contrast, whereas mixed outcomes can produce signed deviations around the group mean. Step 2 uses the posterior expectation of Eq. 20 at the full group size to choose which prompts receive dense groups.

For the smallest partial group,

$$u(2; p) = 2p(1 - p). \tag{21}$$

The numerator of the main paper's priority score, $G\widehat{p}_i(1 - \widehat{p}_i)$, is therefore proportional to the pairwise contrast estimate obtained by substituting $\widehat{p}_i$ for p. With a shared group size, multiplying by G changes the score scale but not the candidate order. Among equal-length candidates, the score therefore favors prompts whose history predicts the greatest pairwise contrast. When expected lengths differ, the denominator trades that contrast potential against response cost.

Why use a plug-in score? Step 2 integrates over the Beta posterior because it compares the utility of a full group before making the primary allocation. Refill uses the lighter plug-in term $\widehat{p}_i(1-\widehat{p}_i)$—obtained by substituting the point estimate $\widehat{p}_i$ for the unknown probability—because it only ranks skipped prompts after the base allocation is fixed. This avoids repeating the full selection calculation. Both scores are small for historically easy or hard prompts and largest near uncertain prompts. A posterior-integrated refill score could also be used, because eligibility, action size, and admission can remain fixed when the priority changes.

Why unseen prompts are ineligible. A symmetric Beta prior has mean $1/2$, which maximizes $\widehat{p}(1 - \widehat{p})$. Treating an unseen prompt as highly uncertain would therefore spend residual capacity before observing any task outcome. CABRA instead makes unseen prompts ineligible, so the uncertainty score applies only after one training observation. The warm-up thus comes from the history rule and not from an added step schedule.

CABRA without history shows what happens otherwise. It treats every under-filled prompt as immediately available and consequently spends residual capacity from the beginning. CABRA makes the same capacity conditional on past outcomes. Comparing their refill rates over training measures the effect of requiring observations before spending on increments.

From contrast to learning value. Binary contrast makes a group-relative update possible but does not determine it. The training reward also contains shaping terms, response likelihoods affect policy gradients, and two mixed-reward groups can produce gradients of different magnitude. Contrast probability is a history-only pre-rollout proxy. Under a purely binary reward, mixed correctness is necessary for a nonzero relative advantage; with shaping, equal correctness outcomes may still receive distinct policy rewards, so that necessity no longer holds.

Group size then controls a second trade-off. Larger groups estimate relative advantages from more responses and can contain richer reward structure, but they spend more tokens on one prompt. A size-two refill accepts a noisier local estimate in exchange for covering a prompt that would otherwise contribute no current group. CABRA leaves the full groups chosen by the base intact, so this trade-off applies only to residual capacity and does not replace the primary learning signal.

Length-aware admission. If every response had equal cost, ranking by contrast potential would be sufficient. Autoregressive responses instead have prompt-dependent lengths. The main paper's priority score discounts expected length by $\sqrt{\ell_i + 1}$, while the admission test charges the full marginal estimate $\Delta\ell_i$. Length is used in two ways. The square-root term is a soft priority that avoids letting small length differences dominate uncertainty; residual admission is a hard feasibility check that prevents an expensive increment from consuming more than the residual allowance.

More generally, one can rank by marginal value per marginal cost. Let p_i be prompt i's response-success probability; then

$$\rho_i(\Delta) = \frac{u(g_i^{(0)} + \Delta; p_i) - u(g_i^{(0)}; p_i)}{\Delta\ell_i}. \tag{22}$$

The current score is a lightweight surrogate for this quantity in the setting where the base allocation gives all or none and refills have size two. Equation 22 also shows how the method extends to several group sizes: each possible addition of responses has its own predicted gain and cost, and the refill pass chooses which additions fit the remaining expected-cost budget.

Three separable decisions. CABRA's incremental step first checks whether a prompt has past outcomes, then ranks eligible increments and tests their expected cost against the remaining budget. With one candidate, eligibility and admission determine the action. With several candidates, changing their priority can also change the accepted set. Either case can occur in an eight-prompt rollout, depending on the base allocation and current history. Because these decisions are separate, each rule can be ablated or replaced while the others and the training objective stay fixed.

A.4 Hyperparameters

The studies use Grade School Math 8K (GSM8K), the Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO) objective, and Quantized Low-Rank Adaptation (QLoRA) with 4-bit NormalFloat (NF4) weights. The Kullback–Leibler (KL) coefficient controls the reference-policy penalty, and the lower and upper clipping thresholds bound the policy probability ratio. Top-p denotes nucleus sampling: generation keeps the smallest highest-probability token set whose cumulative probability reaches p; $p = 1$ applies no probability-mass truncation. Each run uses one NVIDIA A800 graphics processing unit (GPU) with 40 gigabytes (GB) of memory. Qwen2.5-1.5B-Instruct is a 1.5-billion-parameter model.

In the HORA comparison, log probabilities under the behavior policy—the policy that generated the responses—are computed immediately after generation, before either optimizer minibatch updates the policy.

B Additional Baseline Comparisons

B.1 Common-Run Comparison and Additional Runs

The six-allocator comparison uses a common eight-prompt run with seed 46. Additional seeds measure variation within the repeated comparisons. Adapter-backward work is reported in billions of floating-point operations (GFLOPs) per rollout; the $\pm$ notation and "SD" row denote the sample standard deviation. The rows represent dense DAPO, GRESO, hit-utility HORA, adaptive rollout-and-curation AERO, success-predictive VIP, and the proposed CABRA allocation procedure.

Table 3 summarizes the four-seed comparison; the paired measurements below show how accuracy and work vary across seeds.

Table 5: Training settings for the two empirical studies. Seed 42 selects the mechanism-study settings. DAPO, CABRA, HORA, and VIP, the success-predictive allocator, use seeds 43–46 in the eight-prompt study; the Adaptive Efficient Rollout Optimization (AERO) table reports seeds 44 and 46.

Setting	Mechanism study	Eight-prompt study
Base model	Qwen2.5-1.5B-Instruct	
Train / evaluation	GSM8K: 256 train / 1,319 test	
Horizon	300 update steps	75 rollouts, 150 updates
Prompts / dense G	2 / 4	8 / 4
Primary-prompt sampling	independent pool draws	distinct within rollout
Temperature / top-p	1.0 / 0.95	1.0 / 1.0
Learning rate	10^{-5}	
KL coefficient	0.04	0; reference disabled
Policy-ratio clip / high clip	0.20 / 0.28	
Update batching	8-step gradient accumulation	2 disjoint minibatches
QLoRA	4-bit NF4; rank 8, scale 16, dropout 0.05	
Training reward	shaped correctness	± 1 correctness; length penalty
Advantage scale	population SD, floor 10^{-6}	sample SD (correction 1), $\epsilon = 10^{-6}$
Base target / selected	6 / 4 responses	22 / 20 responses
Refill b / integer Q	0.68 / 6	0.68 / 22
Increment / prior	$\Delta = 2$ / Beta(1,1)	
Random seeds	42–46	43–46
Accelerator	one A800 40 GB per run	

Table 6: The seed-46 run used in the main eight-prompt comparison. All rows use Qwen2.5-1.5B, 75 rollouts, 150 updates, and the full GSM8K test set.

Method	Correct / 1319	Accuracy (%)	Backward (GFLOPs/rollout)
DAPO	638	48.37	103.95
GRESO	**643**	**48.75**	104.69
HORA	624	47.31	104.23
AERO	625	47.38	95.17
VIP	626	47.46	104.76
CABRA	630	47.76	**72.00**

CABRA, DAPO, AND HORA

The final-checkpoint accuracy difference is small and changes sign across seeds. Backward work favors CABRA more consistently, with a 22.8% reduction in the four-seed mean. The CABRA summaries record final accuracy and mean backward FLOPs per rollout, but not the full token and wall-time totals.

CABRA AND HORA CONFIGURATION

For each seed, CABRA and HORA use Qwen2.5-1.5B-Instruct, the same prompt sequence, a KL-free DAPO objective (Yu et al., 2025), 75 rollouts, 150 updates, the same optimizer schedule, and the 1,319-example evaluation set. HORA begins with two responses for each of eight prompts and assigns the remaining responses by marginal hit utility, keeping a total response budget of 32. CABRA begins from a base allocation that gives each prompt all or none of its responses and admits a two-response refill only when prompt history is available and the expected-cost budget can accommodate it.

Table 7: Paired HORA comparison. CABRA is more accurate on three seeds and has lower adapter-backward work on all four.

	Accuracy (%)		Backward (GFLOPs/rollout)	
Seed	CABRA	HORA	CABRA	HORA
43	47.23	47.92	73.49	104.77
44	48.52	47.92	104.47	104.74
45	47.23	47.16	72.60	104.10
46	47.76	47.31	72.00	104.23
Mean	47.69	47.57	80.64	104.46
SD	0.61	0.40	15.90	0.35

Table 8: CABRA–HORA differences by seed. Accuracy changes direction across seeds, whereas backward work is lower for CABRA in every pair.

Seed	Correct difference (CABRA − HORA)	Accuracy difference (points)	Backward reduction (%)
43	−9	−0.68	29.86
44	+8	+0.61	0.25
45	+1	+0.08	30.25
46	+6	+0.45	30.93
Mean	+1.5	+0.11	22.80

For seed s, let $c_{m,s}$ be the number of correct answers from method m, and let $f_{m,s}$ be its mean adapter-backward FLOPs per rollout. We report the paired accuracy difference and backward reduction as

$$d_s = 100\,\frac{c_{\mathrm{CABRA},s} - c_{\mathrm{HORA},s}}{1319}, \tag{23}$$

$$r_s = 100\left(1 - \frac{f_{\mathrm{CABRA},s}}{f_{\mathrm{HORA},s}}\right). \tag{24}$$

The ratio of the two four-seed mean costs gives a 22.8% reduction; averaging the four r_s values gives nearly the same result.

Accuracy. The four paired differences range from -0.68 to $+0.61$ percentage points. Three signs favor CABRA, while seed 43 favors HORA. The mean difference is only 0.11 points, or 1.5 answers out of 1,319. The per-seed values show variation that the small mean difference hides.

Backward work. The reduction is close to 30% on seeds 43, 45, and 46. Seed 44 is different: the two methods use almost the same backward work, even though CABRA is more accurate on that pair. Across all four seeds, CABRA never uses more backward work than HORA, and the mean reduction is 22.8%.

Scope of the work metric. Mean backward FLOPs per rollout measures the adapter work contributed by selected responses. It does not include every cost of candidate generation, verification, or scheduling. We therefore call it backward work and not end-to-end training time.

GRESO under the eight-group update. The DAPO update uses eight mixed-reward groups. GRESO searches for those groups under a limit of 4,096 raw candidate rounds and a cap of 32,768 generated responses. In the seed-46 comparison, it forms all eight groups in every rollout, completes 75 rollouts and 150 updates, and answers 643 of 1,319 questions (48.75%) at 104.69 backward GFLOPs per rollout. The run uses 170,456 candidate-prompt draws, 16,416 generated responses,

and 2,400 selected responses. Its reported backward work therefore leaves out a large candidate-generation cost and does not reflect end-to-end training time.

AERO ON SEEDS 44 AND 46

Table 9: On the two listed AERO runs, CABRA is 0.68 percentage points higher on average and uses 7.7% less mean backward work. Bold marks the better value within each seed; the tinted rows mark CABRA.

Seed	Method	Correct / 1319	Accuracy (%)	Backward (GFLOPs/rollout)
Seed 44				
	CABRA	**640**	**48.52**	104.47
	AERO	627	47.54	**96.07**
Seed 46				
	CABRA	**630**	**47.76**	**72.00**
	AERO	625	47.38	95.17
Mean of the two seeds				
	CABRA	**635**	**48.14**	**88.23**
	AERO	626	47.46	95.62

AERO follows adaptive rollout-count and response-curation allocation under the same Qwen2.5-1.5B-Instruct model, prompt sequence, KL-free DAPO objective, 75 rollouts, 150 updates, and 1,319-example evaluation (Zhang et al., 2026). Negative-reward ties use a prompt-seeded rule; the shared-objective comparison does not add AERO's method-specific auxiliary advantage stabilizer.

The accuracy and cost comparisons differ by seed. On seed 44, CABRA answers 13 more questions, while AERO uses 8.0% less backward work. On seed 46, CABRA answers five more questions and uses 24.3% less backward work. The mean row summarizes these two pairs and not the four-seed DAPO–HORA study.

VIP ON SEEDS 43–46

VIP predicts prompt success probabilities and chooses group sizes under a fixed response budget (Nguyen et al., 2026). These runs use the same Qwen2.5-1.5B-Instruct model, prompt sequence, KL-free DAPO objective, 75 rollouts, 150 updates, and 1,319-question evaluation as CABRA. Accuracy differences change sign across seeds, but CABRA uses less backward work in all four pairs. In the common run reported in the main paper, CABRA answers four more questions and reduces backward work from 104.76 to 72.00 GFLOPs per rollout, a 31.3% reduction.

C CONTROLLED AND SEED-LEVEL RESULTS

C.1 MECHANISM-STUDY RESULTS BY SEED

For each comparator, Table 11 reports the mean paired correct-count difference, its 95% confidence interval (CI), and the two-sided paired t-test p-value for a zero mean difference.

C.2 WHERE CABRA HELPS

Comparing CABRA with DAPO measures the work saved after incremental allocation has spent part of the initial saving. On seeds 43–46, CABRA saves 36.4%–38.8% of counted backward work, answers more questions on three of four runs, and matches DAPO's correct-count standard deviation of 1.29. Including seed 42 gives a 38.5% mean cost reduction and four positive accuracy directions in five runs. The $+1.8$-answer paired mean is not statistically significant, while the cost reduction appears on every run.

CABRA w/o refill and VIP use smaller and larger response allocations, respectively. CABRA w/o refill has the largest correct-to-step-work ratio (final correct answers divided by mean GFLOPs per

Table 10: Paired VIP comparison over four runs. Final accuracy differs by only 0.13 points in the mean, while CABRA uses 23.1% less mean adapter-backward work.

	Accuracy (%)		Backward (GFLOPs/rollout)	
Seed	CABRA	VIP	CABRA	VIP
43	47.23	47.31	73.49	105.12
44	48.52	48.07	104.47	105.03
45	47.23	47.38	72.60	104.31
46	47.76	47.46	72.00	104.76
Mean	47.69	47.55	80.64	104.80
SD	0.61	0.35	15.90	0.36

Table 11: Paired five-seed correct-count differences, CABRA minus comparator. All accuracy intervals include zero.

Comparator	Mean difference	95% CI	p-value
DAPO	+1.8	[−0.59,4.19]	0.105
CABRA w/o refill	+1.0	[−6.02,8.02]	0.713
VIP	−1.4	[−5.67,2.87]	0.414
No history	+1.8	[−5.62,9.22]	0.537

step) because it keeps only one full group. On seeds 43–46, CABRA reaches the same mean correct count while forming partial groups on 54.3% of steps and producing an 11.9% higher mean per-step rate of mixed groups per generated token. Compared with the denser VIP allocation, CABRA uses 35.9% less adapter-backward work and gives up 1.75 answers on average. Refill uses more work than skipping without refill but less than global redistribution of the full response budget, with different accuracy trade-offs against each.

CABRA without history shows the effect of timing. Across the four repeats, CABRA uses 10.5% less adapter-backward work, has a 2.5-answer higher mean, and reduces the correct-count standard deviation from 5.89 to 1.29. At the shared $Q = 6$ quota, history gating reduces work at both tested ratios, while the accuracy direction differs: 626 versus 627 correct answers at $b = 0.68$ and 625 versus 620 at $b = 0.75$. History suppresses refill by 82.0 percentage points in the first 50 steps, after which the gap narrows as prompt histories accumulate.

The models also solve different items. Across four seed–model pairs, 72 predictions flip from incorrect under DAPO to correct under CABRA, while 64 flip in the opposite direction. With at most one refill candidate, priority order cannot change allocation volume. One seed-42 random-order control reaches 627 correct at 8.762 GFLOPs, improving on the selected CABRA point on both displayed values; with a single run, this does not rank the methods. CABRA w/o refill gives the largest work reduction, and adding refill spends part of that saving with similar observed accuracy in these runs.

C.3 Additional Cost Statistics

Figure 3 compares adapter-backward work across all five seeds. Table 13 relates that work to final accuracy: each ratio combines the final correct count and mean per-step adapter-backward GFLOPs from one model.

The work-reduction figure and Table 13 show two sides of the trade-off. CABRA's compute reduction varies little across seeds, while final accuracy is more sensitive to the optimization trajectory. The correct-to-step-work ratio favors the most aggressive skip rule; CABRA instead spends additional compute to select partial groups and occupy the region between CABRA w/o refill and DAPO.

Table 12: Exact per-seed adapter-backward work in GFLOPs per step. Table 29 reports the corresponding correct-answer counts. CABRA occupies the intermediate work region on every seed.

Setting	42	43	44	45	46	Mean cut vs. DAPO
DAPO	18.120	17.119	17.202	17.008	17.357	0.0%
VIP	16.868	16.760	16.529	16.646	16.625	3.9%
CABRA without history	11.738	12.107	11.761	11.799	11.961	31.6%
CABRA w/o refill	**8.375**	**8.239**	**8.220**	**8.340**	**8.288**	**52.2%**
CABRA	10.728	10.585	10.534	10.810	10.708	38.5%

Table 13: Final correct answers divided by mean adapter-backward GFLOPs per step, not total-run computation. CABRA raises the mean ratio from 35.8 under DAPO and 52.3 under CABRA without history to 58.4.

Setting	42	43	44	45	46	Mean
DAPO	34.5	36.2	36.1	36.6	35.7	35.8
CABRA w/o refill	74.1	75.9	75.2	75.3	74.7	75.0
VIP	37.1	37.5	37.7	37.4	37.4	37.4
CABRA	58.4	58.8	59.1	57.4	58.3	58.4
CABRA without history	53.4	51.2	53.2	52.7	51.2	52.3

C.4 Accuracy Variation Across Seeds

Table 14 separates seed 42 from seeds 43–46. On the latter, CABRA matches DAPO's correct-count dispersion and is substantially less variable than CABRA without history. Pairwise directions are less uniform: against DAPO, four small gains and one loss across all displayed runs produce a positive mean whose confidence interval still crosses zero. Against CABRA without history, two gains, including a $+12$ result on seed 46, offset three losses.

C.5 Seed-Level Accuracy Differences

The per-seed correct-answer counts in Table 29 differ in pattern across comparators. The DAPO comparison is the most uniform: CABRA gains between one and four answers on four seeds and loses one answer on seed 45. The comparison with CABRA without history looks different. Its $+1.8$ mean includes a $+12$ difference on seed 46, a second $+2$ gain, and three losses. Thus the two rows have the same mean even though their seed-level patterns differ substantially.

The VIP and CABRA w/o refill rows show two further patterns. Two ties against VIP coexist with a negative mean, whereas three five-answer gains against CABRA w/o refill coexist with two losses large enough to leave a mean of only one answer. The paired intervals include zero for all four comparisons, so the five-seed experiment does not separate the settings by average accuracy.

Adapter-backward FLOPs are more consistent. Table 16 subtracts adjacent settings in the two-prompt cost ordering, and every entry is positive. The smallest separation is 0.359 GFLOPs between DAPO and VIP on seed 43; the largest is 5.130 GFLOPs between VIP and CABRA without history on seed 42. In particular, CABRA remains 2.314–2.470 GFLOPs above CABRA w/o refill and 0.989–1.522 GFLOPs below CABRA without history on every seed.

The ordering mirrors the group-size hierarchy: CABRA w/o refill keeps one full group, CABRA adds a history-dependent subset of partial groups, CABRA without history adds them more often, and VIP and DAPO select still more responses. The main paper's adapter-work equation converts these token differences into adapter backpropagation. As a result, CABRA remains between skipping and denser allocation even as response lengths vary across seeds.

Table 14: On seeds 43–46, CABRA matches DAPO's 1.29 correct-count sample standard deviation while shifting the mean from 620.50 to 622.50. The all-five columns additionally include seed 42.

	All five		Repeats 43–46		
Setting	Mean	SD	Mean	SD	Range
DAPO	621.4	2.30	620.50	**1.29**	619–622
CABRA w/o refill	622.2	4.21	622.50	4.80	618–628
VIP	**624.6**	2.88	**624.25**	3.20	622–629
CABRA	623.2	**1.92**	622.50	**1.29**	621–624
CABRA without history	621.4	5.98	620.00	5.89	612–626

Table 15: CABRA exceeds DAPO on four of five paired seeds while using less adapter-backward work on all five. Accuracy differences remain uncertain at five seeds ($p = 0.105$).

CABRA versus	Win	Tie	Loss	Mean difference	p-value
DAPO	4	0	1	+1.8	0.105
CABRA w/o refill	3	0	2	+1.0	0.713
VIP	1	2	2	−1.4	0.414
CABRA without history	2	0	3	+1.8	0.537

D Allocation and Training-Signal Analysis

D.1 Allocation Dynamics

Figure 5 distinguishes delayed eligibility from a globally smaller refill probability. CABRA without history is already active on 96.4% of steps in the first window, while CABRA is active on only 14.4%, an 82.0-point gap. That gap narrows to 46.0, 18.4, and 8.0 points over the next three windows. In the final two windows the settings are separated by only 4.4 and 4.8 points. Most of the aggregate 27.3-point difference therefore arises before enough prompt histories have accumulated, and not from a constant offset maintained throughout training.

The two allocators still differ in the late windows. Table 17 counts 652 partial steps for CABRA and 987 for CABRA without history. The difference follows from history: CABRA refills only prompts seen earlier in training, whereas the control can act immediately. Because a two-prompt batch supplies at most one under-filled candidate, this comparison reveals refill timing and volume; priority ranking becomes relevant only when several candidates compete.

D.2 Rollout Tokens and Backward Compute

The adapter-work equation is linear in selected padded response tokens because all LoRA dimensions are shared. The generated-token totals in Figure 4 produce the same ordering as adapter-backward FLOPs and connect the rising partial-group frequency to the intermediate training cost of CABRA.

Across seeds 43–46, CABRA without history uses 11.7% more rollout tokens than CABRA. Across all five runs, CABRA uses 10.1% less counted adapter-backward work than that control. The small difference between these summaries comes from response-length variation across seeds. Prompt encoding and forward generation add work shared by the same response trajectories but are not part of the main paper's adapter-work equation.

D.3 Prediction Transitions

Table 18 reports how individual predictions change between methods rather than only the net difference in correct-answer counts.

Table 16: CABRA uses more work than CABRA w/o refill and less than CABRA without history on every seed. Entries are adjacent mean per-step adapter-backward FLOP gaps in 10^9; every value is positive, giving CABRA w/o refill $<$ CABRA $<$ CABRA without history $<$ VIP $<$ DAPO.

Seed	CABRA− CABRA w/o refill	No-history CABRA −CABRA	VIP− No-history CABRA	DAPO− VIP
42	2.353	1.010	5.130	1.252
43	2.346	1.522	4.653	0.359
44	2.314	1.227	4.768	0.673
45	2.470	0.989	4.847	0.362
46	2.420	1.253	4.664	0.732
Mean	2.381	1.200	4.812	0.676

Table 17: Allocation patterns over 1200 training steps from seeds 43–46. An allocation $[x, y]$ gives the numbers of responses selected for the two prompts in a step; $[x, y]$ and $[y, x]$ count as the same pattern, and only patterns that occur are listed. The last two columns give the mean numbers of active prompt groups and selected responses per step.

Setting	Allocation	Steps	Active groups	Responses
DAPO	[4,4]	1200	2.000	8.000
CABRA w/o refill	[4,0]	1200	1.000	4.000
VIP	[8,0]	1200	1.000	8.000
CABRA without history	[4,0] / [4,2]	213 / 987	1.823	5.645
CABRA	[4,0] / [4,2]	548 / 652	1.543	5.087

D.4 Two-Prompt Mechanism Study

We use a two-prompt configuration to isolate the decisions inside CABRA. After the base allocation, at most one prompt remains available for refill. Ordering therefore cannot change the allocation, while history eligibility, action size, and budget quantization remain observable. These controls use the seed-42 two-prompt configuration. The repeated two-prompt comparison includes five seeds for CABRA, its ablations, and the external complete-group allocator GRESO. The eight-prompt comparison separately tests allocation when multiple candidates can compete for refill.

Budget quantization and history

With $n = 2$ and $G = 4$, the dense response count is eight, so the tested ratios map to the discrete cells $Q \in \{5, 6, 7\}$.

The complete seven-row budget/history control is reported in main-text Table 4.

At $Q = 5$, changing b from 0.55 to 0.60 leaves backward work unchanged, exposing the budget plateau. The accuracy difference between these rows comes from separately trained policies rather than a different allocation cell. At both tested $Q = 6$ settings, history gating reduces realized backward work, while the direction of the accuracy difference varies. The stable effect of history is thus delayed spending. Small cost differences within one cell arise because sampled responses update histories and length estimates, changing later admissions along each training trajectory.

The integer count Q, the increment Δ, history, and predicted length play distinct roles. Moving to a new Q changes which refill actions can fit; changing b within a cell does not. History determines whether an increment can be proposed, and predicted length determines whether it passes the cap. Budget sweeps should therefore visit distinct Q cells first, then vary eligibility or priority within a cell. Table 4 shows this separation between action capacity and the timing of spending.

Group-size and priority controls

The remaining controls vary either the group-size rule or the order used by the refill pass.

Table 18: Across four seed–model pairs, 72 predictions are correct only under CABRA and 64 only under DAPO. The pooled net is eight, exposing the turnover behind the aggregate difference.

Seed	Both	CABRA only	DAPO only	Net
43	600	22	19	+3
44	608	15	13	+2
45	604	17	18	−1
46	606	18	14	+4
Pooled descriptive	2418	72	64	+8

Table 19: Two-prompt controls isolate group size and ordering. Work is mean adapter-backward GFLOPs per step. Equal backward work within the single-candidate pairs shows that priority cannot change allocation volume when only one prompt is available for refill.

Variant	Correct / 1319	Backward FLOPs (10^9)
Group-size rule		
Posterior group size	618	16.631
Random group sizes	621	13.762
Second-pass refill		
Score order	618	11.237
Random order	623	11.037
Single-candidate controls		
Progress order	623	8.762
Random order	627	8.762
No history, $Q = 5$	624	8.376
Random-order refill, $Q = 5$	624	8.376

For the sole under-filled candidate i, the history-gated decision reduces to

$$g_i = g_i^{(0)} + \Delta\, \mathbf{1}\{N_i > 0\}\, \mathbf{1}\{\Delta\ell_i \le R_0\}\,. \tag{25}$$

Removing history sets the first indicator to one. In either case, the score s_i does not appear: eligibility and expected length determine the action. Equation 25 explains why changing the ordering rule leaves allocation volume unchanged in the last four rows of Table 19.

The group-size controls also separate two ways of reducing work. Choosing a smaller group directly changes the initial allocation, whereas second-pass refill preserves the base allocation and acts only on the capacity left afterward. Similar totals can therefore arise from different sequences of decisions. The latter decomposition is the one used by CABRA: the base continues to decide which prompts receive full groups, and refill has a strictly narrower role.

In the single-candidate state, progress and random order induce the same allocation and hence the same backward work. In the eight-prompt experiment, several eligible prompts can have different predicted values and lengths while the remaining budget admits only a subset. Ranking can then alter which prompts receive refills. Candidate-level diagnostics connect those choices to realized length and reward contrast, separating priority from the more basic effects of eligibility, action size, and admission.

With several candidates, accepting one increment can affect the others. The residual capacity falls by $\Delta\ell_i$, so an early choice can prevent a later candidate from fitting. The score controls that sequence, while the cap still controls total expected work. The two-prompt study isolates the case with no ranking decision; the eight-prompt study executes the scored shared-budget regime in which value and length jointly determine the accepted set. The single-candidate controls test eligibility and action size, but cannot compare ranking formulas when several candidates compete.

Table 20: Group diagnostics separate allocation volume from reward contrast. Fractions are computed from selected groups at each rollout.

Quantity	Definition
Active group	A primary or candidate prompt with at least one selected response.
Partial group	An active group with fewer than four selected responses.
Mixed group	At least two selected responses and a nonzero policy-reward range.
Collapsed group	An active group that is not mixed.
Partial fraction	Partial groups divided by active groups.
Mixed fraction	Mixed groups divided by active groups.

D.5 Reward Contrast and Training Signal

The verifier assigns $+1$ to a correct final numeric answer and -1 otherwise. A shared soft penalty near the response-length limit shapes the policy reward, while CABRA updates its Beta history from final-answer correctness alone. Keeping these quantities separate lets the optimizer respond to length without turning the allocator's success history into a length statistic.

For each active prompt group, the trainer records the range of policy rewards. A group is mixed when it contains at least two responses and its reward range is nonzero; otherwise it is collapsed. This definition matches the group-relative update: subtracting the group mean produces no reward contrast when every reward is equal. Group size and reward contrast remain distinct, because a partial group can be mixed or collapsed just like a full group.

These diagnostics follow the allocation decision through the signal presented to the optimizer. Generated and selected responses measure sampling and filtering; active and partial groups describe allocation structure; mixed and collapsed groups describe reward contrast. Together they show whether an allocator saves work by dropping entire prompts, by shrinking groups, or by selecting groups with little relative signal.

E Experimental Protocol and Reproducibility

E.1 Cost Model

Let $g_i^{(0)} \in \{0, G\}$ be the base allocation and let $\mathcal{P}$ be the set of prompts that receive a refill. The pre-allocation expected token cost decomposes as

$$C(g) = C(g^{(0)}) + \sum_{i \in \mathcal{P}} \delta_i \ell_i, \qquad \delta_i = \min\{\Delta, G - g_i^{(0)}\}. \tag{26}$$

Equation 26 shows how CABRA reuses capacity. The base allocation stays intact, and each additional term is one partial group. The EMA predicts the marginal cost used for allocation, while generated responses determine the realized token count.

The integer Q turns the ratio into an expected-cost budget that changes in steps. Under the main paper's equation, $B = Q\,C_{\text{dense}}/(nG)$, and the allocator accepts increments against $R_0 = \max\{0, B - C(g^{(0)})\}$. It does not separately cap $\sum_i g_i$ at Q: predicted response length determines how many discrete increments fit. The resulting $C(g) \le \max\{C(g^{(0)}), B\}$ invariant preserves the base even when its independent target already places it above B, while a positive residual lets a short partial group use capacity that would not accommodate a longer one.

After generation, the trainer counts selected padded response tokens T and applies the main paper's adapter-work equation. Because adapter rank r, hidden width d, transformer-layer count L, and adapted-projection count M are identical across methods, adapter-backward FLOPs are proportional to T. This proportionality converts changes in selected token counts into the reported backward-work differences. Generation and behavior-policy scoring incur additional costs, so a

Table 21: The cost model links pre-generation allocation to selected tokens. Q and B guide the decision; T determines adapter-backward FLOPs after generation.

Quantity	Meaning and implementation role
n	prompts in the batch; $n = 8$ in the main comparison and $n = 2$ in the controlled study
G	dense responses per prompt; $G = 4$
$g_i^{(0)}$	base allocation for prompt i (all or none)
ℓ_i	pre-allocation response-length EMA for prompt i
$Q = \lceil bnG \rceil$	integer reference count used to form B
B	refill-stage expected token-cost reference from the main equation
δ_i	applied partial increment, at most two responses here
$\mathcal{P}$	prompts selected for a partial refill
T	selected padded response tokens measured after generation
s_i	candidate priority when several skipped prompts can be refilled

given percentage reduction in backward work need not produce the same reduction in end-to-end time.

E.2 Complexity and State

For every prompt key seen during training, CABRA stores two Beta counts, a response-length EMA, and an observation count. For N distinct observed prompts, persistent state is $O(N)$ and updating the selected batch is $O(n)$.

The general per-step allocation computes scores in $O(n)$, sorts at most n candidates in $O(n \log n)$, and performs a single greedy $O(n)$ pass. No auxiliary predictor is trained and no extra model forward or backward pass is added. In our eight-prompt rollouts, the sort operates on at most eight candidates; its cost is small beside autoregressive generation.

Where available, wall time spans the first rollout through the final optimizer update, excluding model loading. It combines allocation, generation, verification, policy scoring, and optimization and is therefore not an allocator-only measurement.

E.3 Implementation and Analysis Details

Shared training code

Within each study, the allocators use the same generator, verifier, and DAPO objective; they differ only in the prompt groups they generate and select. In the HORA comparison, we score the behavior policy immediately after generation, then split each rollout into two disjoint prompt minibatches. Allocation can therefore change generated and selected work without changing the objective.

The policy reward combines answer correctness, $+1$ or -1, with a soft penalty near the response-length limit. CABRA updates its Beta history from correctness alone and tracks response length in a separate EMA. The length penalty can shape policy learning without changing the success history used for allocation.

Evaluation and comparisons

The mechanism study trains seeds 42–46 for 300 update steps; seed 42 selects the settings and seeds 43–46 measure their variation. The HORA comparison begins from the same model revision and prompt sequence for each of seeds 43–46, and trains both methods for 75 rollouts and 150 updates. Every run is followed by greedy decoding on all 1,319 GSM8K test questions. Full-test predictions enable item-level comparisons, while rollout statistics connect group allocation to update cost.

COMPUTE METRIC AND HARDWARE

Every run uses one A800 40 GB accelerator. The mechanism study reports generated and selected padded response tokens and adapter-backward FLOPs. Complete HORA artifacts also record forward tokens and wall time, whereas CABRA's eight-prompt summaries record final accuracy and mean backward FLOPs per rollout. The main comparison therefore uses backward FLOPs, which are recorded for both methods; wall time is not available for both.

E.4 ACROSS-SEED BEHAVIOR

Consistent cost ordering. The five seeds preserve the same cost ordering even though their generated lengths differ. The pattern follows from the action hierarchy: the skip-only setting selects the fewest responses, conditional refill adds a bounded subset, CABRA without history activates that subset more often, and VIP and DAPO select still more responses. Seed variation changes the size of the gaps between methods but not their order. Table 16 reports these gaps for each seed.

Accuracy ordering varies. Accuracy behaves differently. No allocator occupies the top row on every seed: VIP is highest on seed 43, CABRA without history on seed 44, CABRA w/o refill on seed 45, and CABRA is highest among the compared selective policies on seed 46. Allocation changes the training trajectory, but the correct counts remain close and their order varies with the seed.

Pairwise differences. Similar mean differences can come from quite different seed-level patterns. Against DAPO, CABRA produces four small positive differences and one small negative difference. Against CABRA without history, two positive differences, including a $+12$ result, occur alongside three losses. We report mean deltas to measure the average difference, win/tie/loss counts to show how often it occurs, and paired intervals to quantify its uncertainty.

Selected-group work has a consistent ordering across the five runs, whereas accuracy does not. Table 12 gives the exact per-seed work, and Table 29 gives the corresponding absolute correct-answer counts.

E.5 DESIGN TRADE-OFFS

CABRA, its no-refill ablation, and GRESO. CABRA w/o refill removes incremental allocation: it scores the current primary batch, assigns a fixed number of complete groups, and stops. GRESO is the external baseline from Zheng et al. (2025): it uses its own historical prefilter, generates complete groups, removes zero-variance groups, and continues candidate search until the requested mixed groups or a search limit is reached. In contrast, CABRA's incremental step preserves every selected complete group and changes only eligible zero assignments to two-response partial groups.

Table 22: Response allocations and execution rules for CABRA, its no-refill ablation, and GRESO.

Allocator	Action	Defining execution rule
CABRA w/o refill	0 or G	Score the primary prompts once, select the fixed full-group target, then stop.
GRESO	0 or G	Search candidate prompts and select mixed-reward complete groups until it reaches its target or a search limit.
CABRA	0, 2, or G	Preserve complete groups and spend the remaining budget on eligible skipped prompts.

In the eight-prompt common run, GRESO targets eight mixed groups, permits at most 4,096 raw candidate rounds, and caps actual generation at 32,768 responses. The refill interface itself remains composable with any base allocator whose output is $\mathbf{g}^{(0)} \in \{0, G\}^n$, but the reported CABRA experiments use the complete-group rule in Section 2.3.

Table 23: Allocation settings used in the two studies. The corresponding model, reward, optimizer, and evaluation settings appear in Table 5.

Study and allocator	Configuration
Mechanism: DAPO	Two full groups, [4,4].
Mechanism: CABRA w/o refill	Base allocation only, [4,0].
Mechanism: VIP	Redistribute eight responses using posterior success variance.
Mechanism: CABRA without history	Same refill and budget as CABRA, without the history gate.
Mechanism: CABRA	Base allocation followed by size-two refill; actions {0,2,4}, base fraction 0.70, refill ratio 0.68.
HORA study: CABRA	Eight prompts, group sizes {0,2,4}, and the same base and refill ratios.
HORA study: HORA	Two pre-rollouts per prompt, then marginal hit-utility allocation under a 32-response total.

Removing the history condition. CABRA without history keeps the refill but allows it without prior observations. It shows the cost of spending the remaining budget immediately. Partial groups appear from the beginning, their cumulative count grows faster, and selected padded response tokens increase. In the single-candidate regime, this comparison isolates what history contributes to scheduling. The eight-prompt run executes scored multi-candidate allocation; an order counterfactual requires replaying the same candidate states.

Comparison with VIP. VIP predicts prompt success with a Gaussian process, then allocates a fixed budget to minimize predicted gradient variance (Nguyen et al., 2026). Our two-prompt VIP configuration redistributes the same eight-response budget according to posterior success variance. CABRA instead selects complete groups first and adds the smallest useful partial group only when history and remaining cost allow it.

Refill-cell selection. Refill quotas are discrete. Given n and G, enumerate reachable values of Q and choose an increment Δ. Under the main paper's equation, the refill-stage cost reference follows directly from Q and the current dense-length estimate, so ratios that give the same Q behave identically. For a given Q, eligibility, length estimation, and priority determine which feasible increments are admitted.

Multi-candidate execution. With eight prompts per rollout, several skipped prompts can compete for the same residual budget, so eligibility, priority, and admission can affect different decisions. Group-size trajectories show which partial groups are selected. The reported eight-prompt CABRA run executes the main paper's success–length priority before every refill pass; the two-prompt controls deliberately remove ordering from the causal path and isolate eligibility and action size.

E.6 Training and Evaluation Details

Our HORA configuration gives each of the eight prompts two pre-rollouts and assigns the remaining 16 responses by HORA's marginal hit-utility rule, for 32 responses per batch (Wang et al., 2026).

The 256-prompt training pool is sampled independently across 75 rollouts, with eight distinct prompts in each rollout. Repeated prompts build the correctness and length history used by CABRA, while prompts that have not reappeared remain under the base allocation. Across 600 prompt selections, recurrence creates the warm-up without a preset step schedule.

The two-prompt schedules contain the same number of selections: 300 steps times two primary positions gives 600. Table 24 measures recurrence within these training schedules. Across seeds 42–46, 61.3% of selections revisit a prompt that appeared earlier in the same schedule.

For the four confirmation seeds with step-level selected-group traces (43–46), an average of 48.4% of primary prompts are eligible for refill (they have an observed outcome) when allocation is made.

Table 24: Prompt recurrence in the two-prompt training schedules. A revisit is any selection whose prompt key occurred at an earlier position in the schedule.

Seed	Unique prompts / 600	Revisit rate (%)
42	230	61.7
43	235	60.8
44	235	60.8
45	225	62.5
46	235	60.8
Mean	232.0	61.3

Table 25: CABRA dominates DAPO on three of four seeds; only CABRA w/o refill dominates CABRA on any paired run. Deltas are CABRA minus comparator; work is mean GFLOPs per step.

Comparator	Mean Δ correct	Mean Δ FLOPs	CABRA dom.	Trade-off	Control dom.
DAPO	+2.00	−6.513	3	1	0
CABRA w/o refill	0.00	+2.387	0	2	2
VIP	−1.75	−5.981	2	2	0
No history	+2.50	−1.248	2	2	0
Total	–	–	7	7	2

In consecutive 50-step windows, this fraction rises from 9.8% to 28.0%, 44.5%, 60.0%, 70.8%, and 77.3%. The selected groups give at least one stored outcome for 25%, 50%, and 75% of the 256-prompt pool by mean steps 67, 144, and 256, respectively. This matches the observed warm-up, since recurrence builds up history gradually instead of making every prompt eligible at the start.

Both studies evaluate exact final-answer correctness. The mechanism study uses its shaped correctness reward; the HORA comparison uses $+1/-1$ correctness with a shared soft overlong penalty. Evaluation removes shaping and reports only greedy final-answer accuracy. Group diversity therefore describes the signal presented to the optimizer; test accuracy describes the trained policy.

E.7 Paired Accuracy–Cost Outcomes

Table 25 compares accuracy and work jointly without introducing an accuracy or cost threshold. For each of seeds 43–46, CABRA dominates when it is no less accurate, uses no more adapter-backward work, and is strictly better on at least one of the two. Control dominance is defined in the opposite direction; the remaining pairs trade accuracy for cost.

Against DAPO, VIP, and CABRA without history, CABRA sometimes reduces counted cost with equal or higher accuracy and sometimes sacrifices accuracy for that saving. The CABRA w/o refill ablation dominates CABRA twice; in the other two runs, it uses less work but gives fewer correct answers. No single allocation is best on both measures in every run.

E.8 Allocation Examples

The following two-prompt examples illustrate the allocation in Figure 2. The main comparison uses eight prompts per rollout, whereas the controlled mechanism study uses two. Let $n = 2$, $G = 4$, $b = 0.68$, so $Q = \lceil 0.68 \times 8 \rceil = 6$. Suppose the base allocation is $g^{(0)} = [4, 0]$.

Example 1: unseen skipped prompt. The second prompt has no history entry. It is ineligible even though two responses may fit the remaining expected-cost budget. CABRA returns $[4, 0]$ and matches the base allocation. This is why refill is rare before the training pool has been revisited.

Example 2: history exists and expected cost fits. The second prompt has at least one binary history observation, and $2\ell_2 \leq \max\{0, B - C(g^{(0)})\}$. The allocator sets the applied increment $\delta_2 = 2$, returns $[4, 2]$, and the trainer samples six responses in total. The two-response group can produce nonzero standardized advantages if its policy rewards differ. The decision is made before these responses are sampled.

Example 3: history exists but expected cost does not fit. The prompt is eligible, but its length EMA makes the two-response increment exceed the residual expected-cost allowance. The allocation stays $[4, 0]$ even though a count-only rule might admit it. Two configurations with the same Q can therefore realize different token counts.

The symmetric base allocation $[0, 4]$ follows the same traces with prompt indices exchanged. In the evaluated eight-prompt setting, several eligible prompts may compete; the score then orders their increments, and the greedy pass can accept some candidates while rejecting others.

F Scope, Trade-offs, and Extensions

F.1 Failure Modes and Extensions

Cold start for new prompts. Prompt-keyed history is easy to interpret but cannot transfer experience to new or rarely repeated prompts. A shared estimator could pool evidence across semantic clusters, difficulty buckets, or learned features. Comparing prompt-specific and shared histories on a nonrepeating stream would show whether the warm-up depends on recurrence or reflects broader learning progress.

Equivalent budget ratios. Several nominal ratios can map to the same integer cap, so a dense sweep over b may test only a few distinct budgets. Future sweeps should enumerate Q directly, then vary the history rule, cost estimate, or ranking within each quota. If response count and predicted token cost must vary independently, they can be assigned separate controls.

Ranking calibration. The eight-prompt batch creates competition among refill candidates, but end-to-end training couples ranking decisions with the histories they produce. Replaying uncertainty-per-cost, length-only, history-only, and random rankings on the same candidate states would isolate the immediate contribution of the ranking rule.

Task transfer. The same refill rule can be tuned on one split and then applied to a new reasoning domain with a different response-length and reward distribution. Re-estimating only the cost model and quota would test whether the partial action transfers independently of the original task.

Systems realization. Partial groups reduce selected padded response tokens, while wall time also depends on packing, generation, and fixed overhead. A common end-to-end profile across all eight-prompt methods would measure that systems consequence directly.

These extensions target the current limitations: cold start for prompt-specific histories, untested ranking alternatives, evaluation on a single task, and incomplete end-to-end cost measurements.

F.2 Adapting Partial Refill to New Verifiable-Reward Training Settings

Partial refill separates the allocation action from the estimates needed for a particular task. The action remains three-way: *skip*, *add a partial group*, or *allocate a full group*. A base allocator first proposes group sizes from past information; refill then considers under-filled groups, adds increments that fit the resource cap, and passes the resulting groups to the usual verifier and policy update. Adapting CABRA amounts to choosing the base allocation, the smallest useful increment, the history representation, and the cost model.

Base allocation. Let a base allocator return $\mathbf{g}^{(0)} = (g_1^{(0)}, \ldots, g_n^{(0)})$ together with each prompt's full group size G_i. A useful implementation provides four quantities for every under-filled prompt: its current allocation, persistent state key, cost of an additional increment, and any base score used to

form $\mathbf{g}^{(0)}$. The base may be uncertainty-driven, difficulty-driven, learned, or rule-based; refill only requires under-filled groups. This decomposition preserves the behavior that made the base useful while allowing smaller groups for prompts it would skip. For bases that already emit several group sizes, refill can advance a prompt to the next admissible size on the same ordered ladder.

Choosing the partial increment. The increment Δ_i is the smallest group that remains useful to the downstream estimator. Under group-relative normalization, a singleton provides no within-group contrast, whereas the next larger group can already produce a learning signal. The increment can also reflect execution units such as a decoding microbatch (responses decoded together) or a sequence-packing block (variable-length sequences placed in one fixed block). Homogeneous tasks can share one Δ; heterogeneous tasks can choose Δ_i from response type, expected length, or verifier cost. When the dense group is large, a short ladder of group sizes creates several refill levels. What matters is that every accepted action adds a group large enough to provide within-group contrast and uses a measurable amount of residual budget.

Representing history. History supplies a pre-rollout estimate of whether an additional group is informative. Binary verifiers admit a compact Beta state, while bounded continuous rewards can use running means and variances, running quantiles, or a calibrated posterior over reward dispersion. Repeated-prompt training can key this state by prompt identity. In large or nonrepeating streams, the state can instead be keyed by semantic cluster, difficulty bucket, data source, or a learned feature representation. Such shared state could provide estimates for new prompts instead of requiring an earlier observation of the exact same text.

History plays two roles. *Eligibility* decides whether past outcomes are informative enough to spend on a prompt, while *priority* compares eligible prompts. Keeping the roles separate makes the induced warm-up interpretable and allows either component to be replaced independently. For example, a shared history representation can reduce cold start without changing the score, while a new score can improve competition without altering the eligibility threshold.

Cost model and budget. The cost model follows the resource that governs the training loop. Let $c_i(\delta; h_i)$ estimate the marginal cost of adding δ responses to prompt i under state h_i. Token-dominated workloads can use expected generated or selected tokens; verifier-heavy workloads can include scoring cost; distributed training can incorporate padding, communication, or memory pressure. A deployment that also needs a response-slot limit can combine it with an expected resource cap,

$$\sum_i g_i \le Q, \qquad \sum_i c_i(g_i; h_i) \le B. \tag{27}$$

The present refill stage derives B from the integer reference count Q but does not impose the first inequality separately. The second inequality holds when the base is below B; in general the preservation guarantee is $C(\mathbf{g}) \le \max\{C(\mathbf{g}^{(0)}), B\}$, since an over-budget base is returned unchanged. A deployment may keep this step-wise refill budget or expose an independent response-slot limit when tensor shape or scheduler capacity requires one.

The reachable values of Q, and hence of B, are the only distinct budget choices. Changing b without changing Q leaves the budget unchanged. Length estimates update from completed generations, so the same Q values can follow a new model's response-length distribution.

Ranking skipped prompts. When several prompts compete for the remaining budget, priority can take the generic form below, where $\mathrm{value}(h_i, G_i, \Delta_i)$ estimates the learning value of adding increment Δ_i to prompt i with history h_i and full-group size G_i:

$$s_i = \frac{\mathrm{value}(h_i, G_i, \Delta_i)}{c_i(\Delta_i; h_i)}, \tag{28}$$

Suitable choices for $\mathrm{value}(\cdot)$ include predicted reward variance, probability of within-group contrast, posterior uncertainty, or a learned marginal-gain estimate. A ranking decision matters only when multiple eligible candidates share a budget that admits a subset of their increments.

Eligibility decides who may compete, priority decides their order, and action size decides how much each accepted refill adds. Replaying alternative priorities on the same candidate states separates

ranking effects from differences in training volume and shows whether a score favors coverage, shorter responses, or both.

Systems profiling. We currently measure generated and selected padded response tokens, then use the selected tensor shapes to count adapter work. A shared profile of decoding, verification, optimization, communication, memory, throughput, and energy would show where padding or fixed overhead absorbs token savings.

Dynamic batching and sequence packing are especially important. An implementation that compacts partial groups can turn a finer allocation into shorter kernels, whereas a rigid dense layout may preserve much of the original wall-clock cost. Profiling every policy in the same runtime connects discrete allocation decisions directly to systems behavior.

Adapting in practice. Adapting partial refill requires four choices: the base allocation and its remaining candidates, the smallest useful increment, the history representation, and the marginal-cost model. These choices determine which budget levels are reachable and how eligible prompts compete. Latency, memory, and throughput then measure the consequences in the target runtime. The allocator still spends the remaining budget on smaller groups, but its estimates of history, value, and cost have to be adapted to the training setting.

F.3 Other Budget Parameterizations

The refill stage uses a single budget parameter: b selects Q, and Q determines the expected-cost reference through Eq. 16. The base's response fraction ρ is independent. The refill reference moves in average-response units rather than continuously with b. Other systems can parameterize partial-group budgets around their own resource bottlenecks.

Quantized expected-token budget. The current rule sets B to the expected cost of Q average-length responses. Generated length is the resource being controlled; the realized response count may be above or below Q when candidate lengths differ. Rounding to an integer Q aligns the target with average-response units, while the length-weighted admission test prevents predicted long responses from consuming disproportionate work. The budget is therefore specified by (Q, Δ) together with the length estimator, with Q serving as a reference count rather than a second constraint.

Independent count and resource caps. Some systems need separate control over response slots and expected work. In that case, Q and B in Eq. 27 can be independent. Sweeping Q changes the number of admissible actions, while sweeping B at a given Q changes which length profiles fit. This two-dimensional parameterization is useful when memory limits the number of concurrent sequences but latency or energy depends strongly on their lengths. Holding one cap fixed while varying the other allows these constraints to be tested separately.

Budget left after the base allocation. An alternative defines the refill budget relative to the actual base allocation. Let C_{target} be the deployment's expected-cost target; then the capacity left after the base allocation is

$$R_{\mathrm{base}} = C_{\mathrm{target}} - C(\mathbf{g}^{(0)}), \qquad R_{\mathrm{base}} \geq 0. \tag{29}$$

Refill then spends only the capacity left by the base allocation. This is useful when a deployment starts from a fixed cost target, such as the cost of dense training, and reuses the capacity freed by skipping. Complete groups remain unchanged, and the cost of each partial group is subtracted from that capacity.

This residual formulation has two informative regimes. If $R_{\mathrm{base}} < \Delta \min_i \ell_i$, the current step has insufficient capacity for any partial group and CABRA reduces to its base. If R_{base} is large, eligibility or priority rather than capacity becomes the binding decision. How often each case occurs shows whether a chosen budget is actually selective or almost always permissive.

Table 26: The five decisions governing multi-candidate refill.

Component	Role in allocation
Action granularity	Adds a size-Δ option between skipping and a full group.
Eligibility	Delays refill until history supports a prompt-level estimate.
Priority	Orders eligible prompts by predicted value relative to cost.
Admission	Accepts only increments that fit the residual expected-cost allowance.
Cost model	Converts response-length history into the marginal cost used by the allocator.

F.4 Multi-Candidate Allocation

With eight prompts per rollout, several under-filled prompts may share one residual budget. Their allocation can then depend separately on eligibility, score order, cost, and available capacity—distinctions that disappear when only one prompt remains.

When priority affects allocation. Priority matters only when at least two eligible prompts compete for a budget that cannot admit every increment. Batch size alone is insufficient if every candidate fits; likewise, a tight cap has no ranking effect when only one prompt is eligible. These regimes distinguish eligibility and admission from priority.

Immediate and long-term effects. End-to-end training captures how ranking choices alter later histories. Replaying alternative rankings on one candidate state instead isolates the immediate ordering decision, and both views are needed.

Candidate regimes. With one eligible candidate, history and expected cost determine the action, and changing the score cannot change who receives a refill. With two candidates and room for one increment, the score makes a single visible choice. With more candidates, each accepted increment reduces the capacity available to the rest, so early choices can also change how many later prompts fit. As the candidate set grows, the same allocator thus changes from a simple admission rule into a ranking rule over a shared budget.

Separating value from cost. The main paper's priority score combines predicted reward contrast with expected response length. A length-only order tests whether the observed choice is driven by short responses, while a random order removes both signals. Comparing these orders on the same candidate state shows the immediate effect of the numerator and the denominator. Training each order from the same initialization then shows whether those local choices accumulate into different prompt histories and learned policies.

Candidate-level outcomes. The score, predicted marginal cost, admission decision, realized length, and subsequent reward contrast describe a candidate before final accuracy is known. These records distinguish a preference for short prompts from a preference for uncertain prompts. Correlation between predicted value and realized contrast measures calibration; comparison with an oracle order quantifies value left by the greedy choice.

Update-level signals. Candidate choices determine the composition of the policy update. Useful summaries include the number of active prompt groups, distribution of group sizes, fraction of zero-advantage groups, selected padded response tokens, and gradient norm by group size. These metrics relate an allocation decision to the training signal before final evaluation, and they would show if a score selected many groups predicted to be useful that still produced negligible policy gradients.

Training outcomes. Paired accuracy and answer turnover describe the learned policy. Generated and selected padded response tokens describe the allocation; operation counts and wall time describe its execution. Keeping these levels apart prevents attributing a system bottleneck to the ranking score or an accuracy fluctuation to a small cost difference.

Table 27: The same allocation change is visible in cost, training signal, and final predictions. Each row states the principal observation from the corresponding detailed analysis.

Analysis	Main observation
Accuracy and cost	CABRA averages 623.2 correct with a 38.5% backward-FLOP reduction across five seeds.
Prediction turnover	Across four seed–model pairs, 72 predictions are correct only under CABRA and 64 only under DAPO.
Allocation dynamics	CABRA forms a partial group on 652 of 1,200 steps across seeds 43–46.
Budget response	The tested ratios map to discrete quotas $Q \in \{5,6,7\}$ rather than a continuous response.
Reward diversity	CABRA has a 1.165 mean per-step rate of mixed groups per 1,000 generated tokens versus 0.961 for DAPO.

Interpreting interactions. Eligibility and priority can interact. Prompt-specific history may leave few candidates, making their order irrelevant; shared history may admit more candidates and make ranking more important. Smaller increments also spread capacity across more prompts, whereas larger increments make early choices more binding. Ranking therefore matters most when several eligible prompts compete for a limited number of increments.

Direct priority ablation. A direct ranking ablation starts from the same candidate state, with at least two eligible prompts and room for only one increment. Changing eligibility alters who can compete; changing score order alters who receives the refill. Replaying these choices on the same state isolates the immediate action, while retraining from the same seeds captures their effects on later histories and the final policy.

F.5 From Allocation to Learned Behavior

From quota to policy. Complete-group selection reduces the workload, and the quota and history then decide when increments spend part of that saving. The 652 steps with a partial group measure this extra spending. Reward diversity and changes in individual predictions describe the resulting training batches and policy; they do not show that refill improves accuracy.

F.6 Mechanistic Analysis of Allocation Dynamics

The same final response count can arise from different refill schedules, so aggregate compute alone cannot explain how partial groups enter training. To explain these schedules, we separate the state available before generation from the outcomes observed afterward. Table 28 lists the relevant histories, base assignments, refill decisions, rewards, and predictions. Figure 5 uses this distinction to show when prior observations make prompts eligible for partial groups.

Every accepted refill is charged against the nonnegative pre-generation residual. When adapter rank, hidden width, layer count, and adapted projections are shared, the ratio of adapter-work values equals the ratio of padded backward-token totals. These identities link candidate admission to predicted cost and selected groups to the compute axis used in the results.

G Detailed Algorithms and Additional Analyses

G.1 Allocation and training flow

In Algorithm 1 and Figure 2, complete-group selection sets a bounded response count and stops replacement search. These two choices give the direct workload savings. Incremental allocation spends part of the remaining predicted-cost allowance without removing those groups. Its residual starts at $R_0 = \max\{0, B - C(\mathbf{g}^{(0)})\}$ and decreases by $\Delta\ell_i$ after each admitted increment. This allowance limits incremental spending only, not the total realized token cost. All allocations precede the responses that update the next history snapshot.

Table 28: The allocator links historical state to the learned policy through a discrete refill action. The intermediate quantities distinguish when refill occurs, what signal it creates, and how much adapter work it requires.

Stage	Mechanistic quantity
Batch context	sampled prompts and dense group size G
Historical state	Beta counts, observation count, and length EMA ℓ_i
Base allocation	$g_i^{(0)}$, base score or selection reason
Refill action	Q, B, candidate scores, and accepted increments δ_i
Generated outcome	realized group sizes, response lengths, policy rewards, binary history observations
Training signal	reward contrast, selected padded response tokens, and the main paper's adapter-work equation
Learned policy	per-item predictions, extracted answers, and numeric correctness

G.2 Incremental priority and detailed allocation algorithm

For a binary success probability $p \in [0, 1]$, a group of $g \geq 1$ independently sampled responses contains at least one success and one failure with probability

$$u(g; p) = 1 - p^g - (1 - p)^g. \tag{30}$$

This probability is zero for a singleton and positive for $g \geq 2$ whenever $p \in (0, 1)$. At the smallest useful group size, $u(2; p) = 2p(1 - p)$. Therefore the numerator of Eq. 8 satisfies $G\widehat{p}_i(1 - \widehat{p}_i) = \frac{G}{2}u(2; \widehat{p}_i)$: it is exactly a shared scale times the contrast probability obtained by substituting the estimated success rate $\widehat{p}_i$. Among equal-length candidates, the score therefore favors histories near the success boundary rather than prompts that appear consistently easy or hard.

The score predicts contrast; the actual update still depends on shaped rewards and response likelihoods. The score only orders eligible skipped prompts: its numerator predicts size-two contrast, its square-root denominator softly prefers shorter responses, and feasibility remains the separate test $\Delta\ell_i \leq R$. It changes neither eligibility, increment size, nor full groups. The priority requires no extra model pass and can be replaced without changing the refill action, history gate, or cost bound.

Algorithm 2 summarizes the complete pre-generation path. Its history argument is the frozen state $\mathcal{H}_{t-1}$ from Eq. 2; sampling begins only after the adjusted group sizes are returned.

If the base allocator costs $T_{\mathcal{A}_0}(n)$ time, the refill stage costs $O(n \log n)$ time for scoring and sorting and $O(n)$ batch-local working space; its persistent history uses one constant-size record per observed prompt key. The pass invokes no model beyond the policy used after allocation.

G.3 Complete controlled correct-answer counts

Table 29 lists the individual observations used for the controlled means and paired uncertainty estimates.

G.4 History timing and additional allocation diagnostics

If history controls when refills are allowed, the gap to CABRA without history should be largest before prompts have observed outcomes and should shrink as outcomes accumulate. We observe this pattern. Across the five paired runs, CABRA forms a partial group on 54.8% of steps, and its token use lies between that of skipping without refill and that of dense training.

Refill begins at 14.4% in the first window of 50 training steps and exceeds 60% after histories accumulate (Figure 5). The gate thus delays early spending instead of lowering the refill probability uniformly.

Removing history. Removing the history condition preserves the same partial-group action. It is active on 82.1% of steps, compared with 54.8% for CABRA. Across five runs, CABRA consequently uses 10.1% less counted backward work than CABRA without history. History therefore

Algorithm 2 CABRA pre-generation allocation

Require: Prompt batch $\mathbf{x}_t = (x_1, \ldots, x_n)$; frozen history $\mathcal{H}_{t-1}$; base $\mathcal{A}_0$ with outputs in $\{0, G\}^n$; integer full-group size G; integer refill $2 \le \Delta < G$; ratio $b \in [0, 1]$
Ensure: Group sizes $\mathbf{g} \in \{0, \Delta, G\}^n$

```
Read (N_i, p̂_i, ℓ_i) for each x_i from H_{t-1}; for an unseen key use (0, 1/2, 128)
g^(0) ← A_0(x_t, H_{t-1}; G)
g ← g^(0)
Form Q and B by Eq. 6
R ← max{0, B − C(g^(0))}
E ← {i : g_i^(0) = 0 and N_i > 0}
Sort E by decreasing s_i from Eq. 8, breaking ties by decreasing batch index
for all i ∈ E in that order do
    if Δℓ_i ≤ R then
        g_i ← Δ
        R ← R − Δℓ_i
    end if
end for
return g
```

Table 29: Correct answers in the controlled study. All six rows are measured under the same model, 300-step horizon, shaped-correctness objective, and 1,319-question evaluation. Each step starts from two primary prompts with group size four; GRESO may search additional candidates to form its two complete mixed-reward groups.

Method	42	43	44	45	46	Mean
DAPO	625	619	621	622	620	621.4
GRESO	626	620	622	623	621	622.4
VIP	626	629	623	623	622	624.6
CABRA without history	627	620	626	622	612	621.4
CABRA w/o refill	621	625	618	628	619	622.2
CABRA	626	622	623	621	624	623.2

delays residual spending until the allocator has previous outcomes for that prompt; it does not simply lower a fixed refill probability.

Because a two-prompt batch has at most one residual candidate, this comparison isolates eligibility and timing rather than Eq. 8's ordering.

G.5 Additional design conditions

Action and timing. A singleton's reward equals its group mean, so its standardized relative advantage is zero; $\Delta = 2$ is the smallest contrast-capable increment. An unseen prompt remains at the base. After one observation it becomes eligible when cost fits, and Eq. 8 orders competing candidates.

Predicted and realized cost. Allocation reads the length EMA before generation and updates it afterward, so realized tokens may differ. The invariant applies to that pre-generation snapshot; realized lengths enter the next one. Scoring, sorting, and refill cost $O(n)$, $O(n \log n)$, and $O(n)$, with no model or backward pass.

Interpreting the reduction. All eight-prompt methods execute 75 rollouts and 150 updates, so the reported reduction is adapter-backward work per update, not early stopping. In the controlled protocol, it comes with 37.9% fewer generated tokens.

Operating regime and transfer. Partial refill targets recurring prompts for which skipping removes contrast. Its reusable interface is the base allocation, increment, history gate, and cost test; another domain changes the history key and cost calibration, not the trainer interface or cost bound.

G.6 Extended related work

Reasoning and group-based RLVR. Prompting and self-training elicit reasoning traces (Wei et al., 2022; Wang et al., 2022; Zelikman et al., 2022; Lewkowycz et al., 2022); math data and tools broaden them (Hendrycks et al., 2021; Luo et al., 2023; Yu et al., 2023; Gou et al., 2023; Toshniwal et al., 2024; Mitra et al., 2024; Ying et al., 2024; Yang et al., 2024); and outcome or process verifiers score them (Uesato et al., 2022; Lightman et al., 2023; Wang et al., 2023; Cui et al., 2025). GRPO, DAPO, and open reasoning systems turn these signals into group-relative updates (Shao et al., 2024; Yu et al., 2025; Xie et al., 2025; Zeng et al., 2025; Hu et al., 2025), amid saturation and uneven token-level signal (Liu et al., 2025; Wang et al., 2025). CABRA keeps the verifier, objective, and update fixed and changes response counts.

Post-training and execution. RLHF and preference optimization define the post-training layer (Ouyang et al., 2022; Rafailov et al., 2023; Lambert et al., 2024). Our experiments use QLoRA adaptation (Hu et al., 2022; Dettmers et al., 2023) and hold scalable RLHF execution frameworks fixed (Hu et al., 2024; Sheng et al., 2024). CABRA changes rollout allocation only.

Selection and rollout allocation. Data and token selection reduce low-value work (Xie et al., 2023; Xia et al., 2024; Lin et al., 2024). GRESO and HIVE choose prompts for full groups (Zheng et al., 2025; Wu et al., 2026). HORA reallocates after same-step evidence; VIP globally chooses group sizes; AERO couples counts with curation; and CERO and DynaMO alter cross-epoch or advantage allocation (Wang et al., 2026; Nguyen et al., 2026; Zhang et al., 2026; Zong et al., 2026; Fang et al., 2026). Contextual Rollout Bandits learns history-based scheduling (Lu et al., 2026). CABRA's refill step keeps the selected groups of G responses, reads only state from earlier rollouts, and adds $0 \to \Delta$ refills within the remaining expected-cost budget. The complete-group step provides the main cost-aware selection. Methods that also change the prompt stream, generation topology, or advantage rule are related mechanisms rather than rows in the matched allocation table.

Generation topology and credit assignment. LATR and Tree-GRPO use lookahead or shared-prefix trees (Xing et al., 2025; Ji et al., 2025); SRPO routes collected samples; SALT changes step-level credit; and FEST changes generation context (Li et al., 2026; 2025; Yan et al., 2026). CABRA acts earlier: it sets response counts, then leaves generation, verification, reward, loss, and optimization unchanged.

G.7 Additional accuracy and signal diagnostics

Across five paired runs, correct-answer differences remain small while CABRA reduces work against every denser comparator. In the common run, CABRA answers 624 of 1,319 questions at 10.71 GFLOPs of adapter-backward work per step. DAPO answers 620 at 17.36 GFLOPs, so CABRA answers four more questions on this run with 38.3% less backward work.

Table 29 makes the accuracy scale explicit. The five-seed means span 621.4–624.6 correct answers, only 3.2 questions or 0.24% of the evaluation set. CABRA averages 623.2 correct answers: 1.8 more than DAPO and CABRA without history, 1.0 more than CABRA w/o refill, 0.8 more than GRESO, and 1.4 fewer than VIP. Even the largest within-seed spread across all six methods is 12 answers (0.91%). These outcomes are close, but we have tested neither equivalence nor an accuracy gain. The no-refill ablation and full method occupy different work levels.

Figure 3 shows the cost decomposition: CABRA spends more than its no-refill ablation to generate partial groups, yet less than each measured denser allocator. It reduces backward work by 38.5% relative to DAPO, 39.0% relative to GRESO, 36.0% relative to VIP, and 10.1% relative to CABRA without history on average. It uses 28.7% more work than CABRA w/o refill because it spends residual capacity on a partial second group. In controlled seed 46, that increment costs 2.42 GFLOPs and answers five more questions than CABRA w/o refill; DAPO spends a further 6.65 GFLOPs.

Table 30: Signal obtained from selected-group work over four confirmation runs. Mixed groups have at least two responses with nonzero policy-reward range. Correct / (GFLOPs/step) divides final correct answers by mean per-step adapter-backward work, not total-run computation; "1k tokens" means 1,000 generated response tokens. Relative changes use unrounded means.

Quantity	DAPO	CABRA	CABRA vs. DAPO
Mixed groups / 1k tokens	0.961	**1.165**	+21.2%
Correct / (GFLOPs/step)	36.2	**58.4**	+61.6%

HORA is identical to DAPO in this two-prompt budget because both allocate $[4, 4]$, and AERO is evaluated only in the compatible eight-prompt protocol in Table 2.

Generated tokens and measured reward contrast. For the full method, Figure 4 first checks the sampled volume: CABRA generates 37.9% fewer rollout tokens than DAPO, 90.9% fewer than GRESO, and 10.5% fewer than CABRA without history across four confirmation seeds. Table 30 then tests whether the selected-group work carries training signal.

The measured token-normalized contrast is higher: CABRA presents 21.2% more mixed groups per thousand tokens and a 61.6% higher correct-to-step-work ratio. Across the same four pairs, 72 run–item predictions are correct only under CABRA and 64 only under DAPO, so cheaper updates are not associated with a strict subset of dense correct predictions. CABRA w/o refill, which uses only the base allocation, remains the variant with the least work.

Protocol roles and budget steps. The two protocols serve different purposes. With eight prompts, multi-candidate scoring is active; with two, ordering has no effect, which isolates the refill and the history gate. Because Q is rounded, the ratio acts in steps: for $nG = 8$, $b \in \{0.55, 0.60\}$ gives $Q = 5$, while $b \in \{0.68, 0.75\}$ gives $Q = 6$. Thus Q, Δ, eligibility, and the frozen length snapshot are the operative controls; Appendices C and D report per-seed cost bands, ordering controls, and reward diversity.