RANKBUFFER: EFFICIENT RANKING-BASED REWARDS FOR OPEN-ENDED GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Open-ended generation lacks canonical answers, making pointwise rewards difficult to calibrate for group-based reinforcement learning. Directly ranking same-query rollouts provides a more suitable relative reward signal, but existing ranking-based reward methods can incur substantial judging cost. We introduce **RankBuffer**, which maintains an ordered, query-specific buffer of previously judged responses as a reusable quality scale. Each rollout is first inserted into an anchor interval through an independent coarse judgment, after which only rollouts assigned to the same interval undergo local fine ranking. The resulting complete order is converted into bounded rank rewards, while boundary expansion, local refinement, and inactive-anchor pruning adapt the buffer as the policy evolves. Across four open-ended benchmarks, RankBuffer consistently outperforms all pointwise baselines. It also achieves nearly on-par performance with the strongest ranking-based reward baseline while substantially reducing judging cost. Ablations demonstrate the importance of both local fine ranking and anchor response content, while buffer analyses show that rollout-derived anchors progressively extend and refine the covered quality scale. These results establish response reuse as an effective approach to efficient relative reward construction. [1]

1 INTRODUCTION

Reinforcement learning (RL) has become an important approach for improving the reasoning and instruction-following capabilities of large language models (LLMs) (Ouyang et al., 2022; Shao et al., 2024; Yu et al., 2026). Group-based algorithms such as Group Relative Policy Optimization (GRPO) (Shao et al., 2024) have emerged as a widely adopted optimization paradigm: instead of relying on a learned value function, GRPO samples a group of rollouts for each query and estimates the advantage of each rollout relative to others within the same group. For tasks with verifiable outcomes, such as mathematics and code generation, rewards can often be obtained from exact-match rules, unit tests, or programmatic verifiers (Shao et al., 2024; Guo et al., 2025). In open-ended generation, however, multiple responses may be valid and no unique reference answer is available. Reward construction for these tasks therefore commonly relies on an LLM judge equipped with query-specific rubrics (Gunjal et al., 2026; Huang et al., 2025; Wei et al., 2026).

Most existing rubric-as-reward methods ask the judge to assign a pointwise reward to each rollout independently (Shen et al., 2026; Xu et al., 2026b). However, such pointwise rewards only provide an indirect and often unstable reward signal for open-ended generation. The challenge is twofold. First, without a canonical answer, the judge must assign pointwise rewards to complex responses on a numerical scale that is difficult to calibrate, and the resulting rewards can vary across queries, rubrics, and training stages (Zheng et al., 2023; Dubois et al., 2024). Second, GRPO relies on relative differences within a rollout group to estimate advantages, whereas independently assigned pointwise rewards may fail to capture these differences reliably: rewards within a group often cluster on a narrow scale, yielding vanishing or noisy advantages precisely where discriminative signal is most needed.

To address these issues and stabilize group-based RL training, recent studies construct rewards from relative comparisons among rollouts rather than independent pointwise rewards. ArenaRL (Zhang

[1] Code and reproduction instructions are available in an anonymous repository at `https://anonymous.4open.science/r/review-artifact-2027-5FCD/`.

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.

et al., 2026b) combines reference-based pre-ranking with a seeded elimination tournament; RRC-AGR and RRC-SCR (Wang et al., 2026) derive rewards from pairwise wins against reference responses or among rollouts within the same group; Tournament-GRPO (Yang et al., 2026b) builds relative rewards through repeated multi-round tournaments. While these methods provide ranking signals that are better suited to group-relative advantage estimation, they incur substantial judging cost: each new rollout group requires multiple comparison or ranking calls, tournament rounds must be executed sequentially, and previously judged responses are discarded rather than reused. As a result, existing ranking-based reward methods trade away the efficiency of pointwise reward methods.

In this work, we ask whether ranking-based reward signals can be obtained at near-pointwise judging cost. We introduce **RankBuffer**, a buffer-assisted coarse-to-fine ranking framework for open-ended generation. RankBuffer maintains an ordered, query-specific buffer of previously judged responses as a reusable quality scale. Each new rollout is inserted into an anchor interval through a single independent judgment—incurring a per-rollout judging pattern comparable to that of pointwise reward methods and fully parallelizable—and only rollouts assigned to the same interval undergo local fine ranking. Concatenating the local orders yields a complete group ranking, which is converted into bounded rank rewards for GRPO advantage estimation. In this way, RankBuffer amortizes comparison costs across training iterations: judgments made for earlier rollout groups are retained as anchors rather than discarded, so that relative rewards are constructed from accumulated evidence instead of rebuilt from scratch for every group. To keep the buffer aligned with the evolving policy, we further design rule-based maintenance through boundary expansion, local refinement, and inactive-anchor pruning, which progressively extends and refines the covered quality scale. Figure 1 contrasts RankBuffer with pointwise reward methods and conventional ranking-based reward methods.

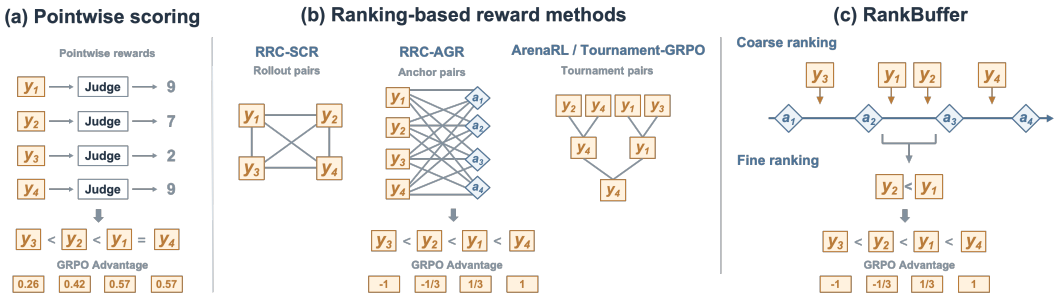


Figure 1: Reward construction with pointwise reward methods, conventional ranking-based reward methods, and RankBuffer. Pointwise rewards are assigned independently, so reward values within a rollout group may cluster or tie and obscure relative differences. Ranking-based reward methods first obtain a complete order and map it to evenly spaced reward levels that preserve the relative quality of the rollouts. RankBuffer produces the same structured reward signal using reusable anchors and selective fine ranking.

Our contributions are summarized as follows:

- We introduce **RankBuffer**, a buffer-assisted coarse-to-fine ranking framework for open-ended generation. It reuses an ordered anchor buffer across iterations and combines independent insertion with selective local fine ranking to construct rewards for GRPO.
- We design rule-based buffer maintenance through boundary expansion, local refinement, and inactive-anchor pruning. We study Nectar and bootstrap initialization and compare dynamic maintenance with static and LLM-based alternatives.
- Across four benchmarks, RankBuffer improves the strongest pointwise baseline by 3.83 average points and achieves comparable performance to other ranking-based reward methods while substantially reducing judging cost. Further analyses examine anchor count, the role of fine ranking, and buffer evolution.

2 Related Work

2.1 Rubric-Guided Judging and Rewards

Early LLM evaluators introduced explicit rubrics and rubric-guided judges (Liu et al., 2023; Kim et al., 2024; Zhu et al., 2025). Recent work develops reasoning reward models, refined rubrics, harder reward-model evaluation, and localized evidence-based adjudication (Chen et al., 2026; Shen et al., 2026; Malik et al., 2026; Yang et al., 2026a). Nevertheless, LLM judgments remain sensitive to position and verbosity (Zheng et al., 2023; Dubois et al., 2024).

Preference-based RL and AI feedback underpin language-model alignment (Christiano et al., 2017; Stiennon et al., 2020; Ouyang et al., 2022; Bai et al., 2022). Recent methods extend RL to open-ended tasks with query-specific rubrics (Gunjal et al., 2026; Huang et al., 2025; Wei et al., 2026). Despite their variety, these methods use pointwise rewards: each rollout is evaluated independently against its rubric and mapped to a scalar reward. Other work adapts evaluators through learned rubric generation, pairwise comparison, or process supervision (Xu et al., 2026b; Jia et al., 2026; Ding et al., 2026; Zhang et al., 2026a). These reward signals can be used with GRPO (Shao et al., 2024) and subsequent group-based objectives (Liu et al., 2024; Yu et al., 2025; Chu et al., 2026; Liu et al., 2026).

2.2 Ranking-Based Reward Methods

For retrieval, LLMs have been used to rank candidates through pairwise comparisons and other efficient ranking strategies (Qin et al., 2024; Chen et al., 2025; Liu et al., 2025a;b). Ranking-based reward methods for RL instead compare policy rollouts through tournaments, reference-guided competition, or pairwise rewards (Yang et al., 2026b; Zhang et al., 2026b; Wang et al., 2026). RankBuffer differs from these methods by maintaining a persistent ordered response scale with an off-the-shelf judge. Each new rollout is inserted relative to existing anchors to identify its interval, and only rollouts within the same interval require further comparison. This allows RankBuffer to reuse prior judgments without training a specialized reward model or reconstructing comparisons for every rollout group.

3 Method

3.1 Overview

Figure 2 summarizes the ranking and buffer-maintenance workflow. Pseudocode is provided in Appendix A.

Given a query x and its query-specific weighted rubrics $\mathcal{R}_x$, the policy generates a group of G rollouts:

$$\mathcal{Y}_x^t = \{y_1^t, \ldots, y_G^t\}, \qquad y_i^t \sim \pi_{\theta_t}(\cdot \mid x), \tag{1}$$

where t denotes the current training iteration and $\pi_{\theta_t}(\cdot \mid x)$ is the response distribution conditioned on query x under the current policy with parameters θ_t.

RankBuffer maintains a query-specific buffer of anchor responses:

$$\mathcal{A}_x^t = (a_1^t, \ldots, a_{K_t}^t), \qquad a_1^t \prec \cdots \prec a_{K_t}^t, \tag{2}$$

where a_k^t is the k-th anchor response stored for query x at iteration t, K_t is the number of anchors in this buffer, and $\prec$ denotes the stored quality order from worst to best.

RankBuffer performs coarse ranking followed by selective fine ranking, maps the resulting group order to rewards, and updates the buffer for subsequent training iterations.

3.2 Buffer Initialization

We consider two ways to construct the initial query-specific buffer.

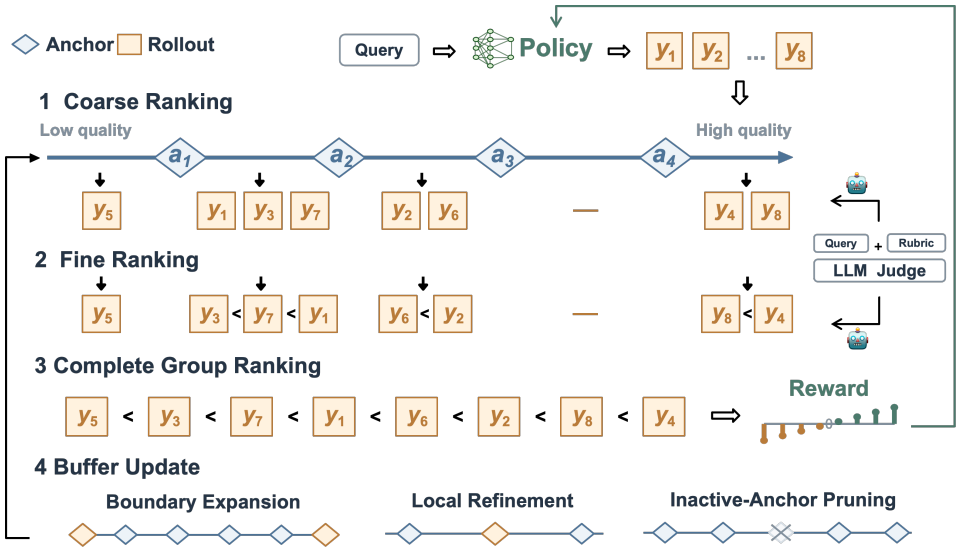


Figure 2: RankBuffer workflow. An ordered, query-specific buffer enables parallel coarse ranking, followed by local fine ranking within intervals containing multiple rollouts. The resulting complete ranking determines the rewards and GRPO advantages. Boundary expansion, local refinement, and inactive-anchor pruning update the buffer for subsequent iterations.

Nectar initialization. For **RankBuffer**, we select K_0 responses from the ranked responses associated with query x in Nectar (Zhu et al., 2024). The selected anchors are stored from worst to best:

$$\mathcal{A}_x^0 = (a_1^0, \ldots, a_{K_0}^0), \tag{3}$$

where K_0 denotes the initial buffer size.

Bootstrap initialization. For **RankBuffer-Bootstrap**, we use listwise judging during the first training epoch to rank the initial rollout group for each query x from worst to best:

$$\sigma_x^0 = J\left(x, \mathcal{R}_x, \mathcal{Y}_x^0\right), \tag{4}$$

where J denotes the LLM judge and σ_x^0 is the resulting worst-to-best rollout ordering. We then select K_0 rollouts from this ordering as anchors to initialize the buffer:

$$\mathcal{A}_x^0 = \mathrm{Select}_{K_0}\left(\sigma_x^0\right). \tag{5}$$

3.3 Buffer-Assisted Coarse-to-Fine Ranking

Coarse ranking. The K_t ordered anchors divide the response-quality space into $K_t + 1$ intervals. For each rollout y_i^t, the judge predicts its insertion position:

$$p_i^t = J\left(x, \mathcal{R}_x, \mathcal{A}_x^t, y_i^t\right) \in \{0, \ldots, K_t\}. \tag{6}$$

Position $p_i^t = 0$ places the rollout below the lowest anchor, while $p_i^t = K_t$ places it above the highest anchor. An internal position p_i^t places the rollout between $a_{p_i^t}^t$ and $a_{p_i^t+1}^t$.

The insertion positions partition the rollout group into buckets:

$$\mathcal{C}_p^t = \{y_i^t \mid p_i^t = p\}, \qquad p \in \{0, \ldots, K_t\}. \tag{7}$$

Here, $\mathcal{C}_p^t$ contains all rollouts with insertion position p at iteration t.

All insertion judgments are performed independently and can be executed in parallel.

Fine ranking. For every bucket containing at least two rollouts, i.e., $|\mathcal{C}_p^t| \geq 2$, we perform listwise ranking within the bucket. The judge returns a permutation of the candidate IDs from worst to best:

$$\sigma_p^t = J\left(x, \mathcal{R}_x, \mathcal{C}_p^t\right). \tag{8}$$

Here, σ_p^t denotes the local rollout order for bucket p. The final group ordering first follows the insertion positions p_i^t in ascending order, with rollouts sharing a position ordered by the fine-ranking result. Single-rollout buckets require no fine ranking. We denote the resulting complete worst-to-best ordering by σ^t. The coarse- and fine-ranking prompts are provided in Appendix B.

3.4 Rank-Based Reward and Advantage Estimation

Let $q_i^t \in \{0, \ldots, G-1\}$ denote the position of rollout y_i^t in the final worst-to-best ordering σ^t. We map this position linearly to a rank reward:

$$r_{i,t}^{\text{rank}} = -1 + \frac{2q_i^t}{G-1}. \tag{9}$$

The training reward additionally includes an auxiliary format reward:

$$r_{i,t} = r_{i,t}^{\text{rank}} + \lambda_{\text{fmt}} r_{i,t}^{\text{fmt}}, \tag{10}$$

where $r_{i,t}^{\text{fmt}}$ indicates whether the rollout follows the required output format.

GRPO normalizes the rewards within each rollout group to estimate the advantages:

$$\widehat{A}_{i,t} = \frac{r_{i,t} - \overline{r}_t}{\sqrt{\frac{1}{G-1}\sum_{j=1}^{G}\left(r_{j,t} - \overline{r}_t\right)^2 + \epsilon}}, \tag{11}$$

where

$$\overline{r}_t = \frac{1}{G}\sum_{j=1}^{G} r_{j,t}. \tag{12}$$

The estimated advantages are then used in the standard GRPO policy objective.

3.5 Rule-Based Buffer Update

After ranking the current rollout group, we update the query-specific buffer using boundary expansion, local refinement, and inactive-anchor pruning.

Boundary expansion. If one or more rollouts are assigned below the lowest anchor, we add the lowest-ranked rollout in $\mathcal{C}_0^t$ to the beginning of the buffer. Similarly, if one or more rollouts are assigned above the highest anchor, we add the highest-ranked rollout in $\mathcal{C}_{K_t}^t$ to the end of the buffer. At most one response is added at each boundary for a query in one epoch.

Local refinement. For an internal bucket $\mathcal{C}_p^t$, where $p \in \{1, \ldots, K_t - 1\}$, if $|\mathcal{C}_p^t| \geq 3$, we add its lower-median rollout as a new anchor:

$$a_{\text{new}} = \sigma_p^t\left[\left\lceil \frac{|\mathcal{C}_p^t|}{2} \right\rceil\right], \qquad |\mathcal{C}_p^t| \geq 3. \tag{13}$$

The new anchor is inserted between the two anchors defining that interval.

Inactive-anchor pruning. For each internal anchor a_k^t, where $k \in \{2, \ldots, K_t - 1\}$, we track whether either of its adjacent intervals receives a rollout. The inactivity counter h_k is initialized to zero when the anchor is created and updated once per epoch as

$$h_k^t = \begin{cases} 0, & \mathcal{C}_{k-1}^t \cup \mathcal{C}_k^t \neq \varnothing, \\ h_k^{t-1} + 1, & \text{otherwise.} \end{cases} \tag{14}$$

An internal anchor is removed after being inactive for H consecutive epochs. The lowest and highest anchors are never pruned. The resulting ordered buffer becomes $\mathcal{A}_x^{t+1}$ and is used to rank future rollout groups.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Training framework and models. We build our RL pipeline on top of verl[2] and optimize the policy with GRPO (Shao et al., 2024). The policy is Qwen3-8B (Yang et al., 2025), and the LLM judge is a locally deployed Qwen3-30B-A3B-Instruct-2507 (Yang et al., 2025). Unless specified otherwise, all methods use the same policy, judge, and number of training steps. We sample $G = 8$ rollouts per query and report results after four training epochs (200 steps). Detailed hyperparameters and hardware settings are provided in Appendix C.

Training data and buffer initialization. Following the two initialization strategies in Section 3.2, Nectar initialization uses 1,000 queries randomly sampled from Nectar (Zhu et al., 2024) with seed 42, selecting four initial anchors per query at Nectar ranks $(7, 5, 3, 1)$. Bootstrap initialization uses the same 1,000 queries but constructs its anchors from policy rollouts instead of Nectar responses: during the first epoch, each query's initial rollout group is ranked listwise using the prompt in Appendix B.2, and four rollouts at worst-to-best positions $(1, 3, 6, 8)$ are selected as anchors.

Query-specific rubrics. Each query is paired with three to five weighted, query-specific rubrics generated once by DeepSeek-V4-Flash (Xu et al., 2026a) and then held fixed. Rubric-generation details and prompts are provided in Appendix D.

Evaluation. We evaluate on AlpacaEval 2 (Dubois et al., 2024), Arena-Hard v2 (Li et al., 2024), WildBench v2 (Lin et al., 2025), and WritingBench (Wu et al., 2026). We use Qwen3-30B-A3B-Instruct-2507 (Yang et al., 2025) as the judge for all four benchmarks.

4.2 REWARD QUALITY AND JUDGING EFFICIENCY

RankBuffer uses Nectar initialization (Section 3.2) with rule-based updates (Section 3.5). **RankBuffer-Bootstrap** uses bootstrap initialization (Section 3.2) and the same rule-based buffer updates as RankBuffer. Implementation details are provided in Appendix E. We compare RankBuffer with two categories of baselines:

Pointwise reward methods. These baselines assign pointwise rewards to rollouts independently using the query-specific rubrics. We include GRPO, GDPO (Liu et al., 2026), Dr. GRPO (Liu et al., 2025c), DAPO (Yu et al., 2025), and GPG (Chu et al., 2026). Table 1 reports the results. RankBuffer outperforms all pointwise baselines on all four benchmarks.

Table 1: Comparison with pointwise baselines.

Method	AE2	AH-v2	WB-v2	WrB
Base	31.11	9.30	37.23	62.49
GRPO	72.73	<u>27.50</u>	46.57	65.42
GDPO	64.35	20.10	45.14	64.94
Dr. GRPO	56.15	18.90	44.32	63.37
DAPO	<u>74.22</u>	26.60	<u>46.83</u>	<u>66.13</u>
GPG	53.04	16.90	45.78	64.98
RankBuffer	**82.80**	**30.20**	**48.79**	**67.27**

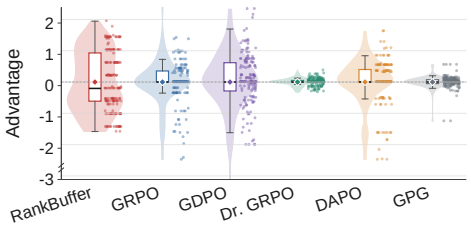


Figure 3: Advantages at step 191.

Figure 3 shows the normalized advantages at training step 191. RankBuffer converts rollout rankings into distinct advantage levels, providing a richer signal for within-group optimization. In contrast, pointwise reward baselines often produce clustered advantages, offering less differentiation among rollouts.

[2] https://github.com/verl-project/verl

Ranking-based reward methods. We include Tournament-GRPO (Yang et al., 2026b), RRC-AGR and RRC-SCR (Wang et al., 2026), Pairwise Rank (exhaustive pairwise judging followed by win-rate ranking; Appendix E), and ArenaRL (Zhang et al., 2026b). These baselines construct rewards through relative comparisons rather than pointwise rewards. Figures 4 and 5 compare average scores against total judge input and output tokens and successfully parsed judge requests during RL training, respectively. RankBuffer achieves comparable or better performance than the ranking-based reward baselines while substantially reducing judge-token cost. It achieves nearly the same performance as the strongest ranking-based reward baseline, RRC-SCR, while using less than half its judge tokens. Full results are provided in Appendix F.

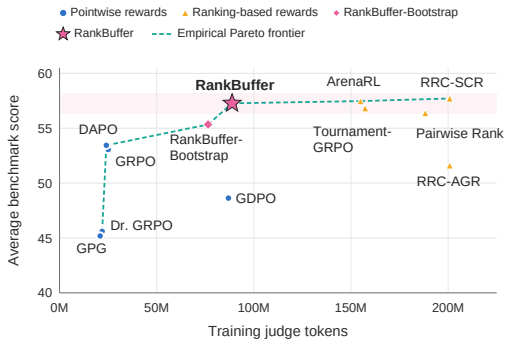


Figure 4: Average score versus judge tokens.

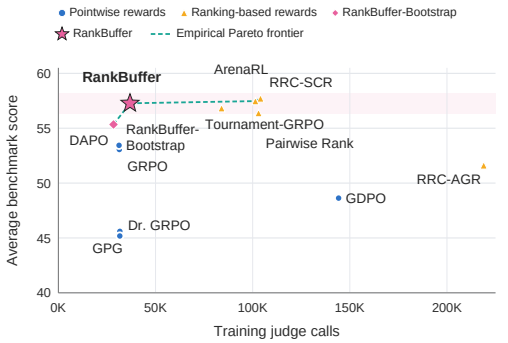


Figure 5: Average score versus judge calls.

4.3 Buffer Design Analysis

Initialization and dynamic maintenance. RankBuffer* and RankBuffer-Bootstrap* freeze their buffers at the end of the first epoch and perform no further buffer updates. Table 2 compares these four configurations. RankBuffer achieves the highest average score (57.27), outperforming RankBuffer* by 2.26 points and demonstrating the benefit of adapting Nectar-initialized anchors during training. For bootstrap initialization, RankBuffer-Bootstrap* yields a higher average score and lower judge cost than RankBuffer-Bootstrap, suggesting that policy-generated initial anchors can also provide a useful fixed reference.

Table 2: Effect of buffer initialization and maintenance. An asterisk (*) denotes static buffers.

Method	AE2	AH-v2	WB-v2	Writing	Avg.	Tokens(M)	Calls(K)
RankBuffer	82.80	30.20	48.79	67.27	**57.27**	88.8M	36.9K
RankBuffer*	79.69	26.70	46.95	66.71	55.01	68.0M	36.2K
RankBuffer-Bootstrap	79.50	28.40	47.31	66.15	55.34	76.5M	28.4K
RankBuffer-Bootstrap*	81.61	29.60	47.47	67.48	**56.54**	60.7M	28.1K

LLM buffer variants. We additionally test whether the buffer-update rules can be replaced by LLM. After each ranking step, LLM receives the latest combined worst-to-best ordering of the current anchors and newly ranked rollouts, and selects four responses from this ordering as the next-step anchors. A second variant, LLM + look-ahead, additionally provides LLM with the next epoch's unranked rollouts as context. The detailed update procedures and prompts are provided in Appendix G.

As shown in Table 3, LLM reaches an average score of 56.60, which is 0.67 points below RankBuffer, while using 8.7% more judge tokens and 12.8% more calls. LLM + look-ahead achieves a lower average score (54.98) than LLM (56.60), indicating that providing the next epoch's

unranked rollouts as context for buffer updates does not improve downstream performance in this setting.

Table 3: Comparison of RankBuffer, RankBuffer-Bootstrap, and LLM-based buffer management.

Method	AE2	AH-v2	WB-v2	Writing	Avg.	Tokens(M)	Calls(K)
RankBuffer	**82.80**	30.20	**48.79**	**67.27**	**57.27**	88.8M	36.9K
RankBuffer-Bootstrap	79.50	28.40	47.31	66.15	55.34	76.5M	28.4K
LLM	81.74	**30.50**	47.34	66.83	56.60	96.5M	41.6K
LLM + look-ahead	79.07	28.40	46.31	66.13	54.98	111.0M	42.1K

4.4 How Does the Dynamic Buffer Evolve?

Figure 6(a) tracks the source of every live anchor at the end of each epoch. Starting from four Nectar anchors per query, the mean buffer size grows to 5.44 after epoch 0 and then to 6.78, 7.27, and 7.48. The mean number of Nectar anchors declines, while right-boundary expansion and local-refinement anchors increase substantially. Left-boundary expansion anchors also increase, but remain a small component of the buffer. The buffer thus progressively incorporates policy-generated anchors, primarily through upper-boundary expansion and interior refinement. Growth slows over time as the removal of inactive anchors offsets the insertion of new rollout anchors. The buffer thus adapts to policy changes while exhibiting progressively smaller size increases over the observed training horizon.

Figure 6(b) shows anchor sources along the quality scale. To compare different-length buffers, we normalize anchor ranks to 0–100 and plot source proportions in ten bins at each epoch end. Initial Nectar anchors shift toward lower normalized positions, while local-refinement anchors concentrate in higher-quality interior regions. Right-boundary expansion anchors dominate the highest-quality bin, increasing from 71% in epoch 0 to 82% in epoch 3. This trend is consistent with progressive improvements in overall rollout quality during training, with responses extending beyond the original buffer's upper quality boundary. Together, these analyses show that buffer updates extend the covered quality range and improve ranking resolution where current rollouts concentrate.

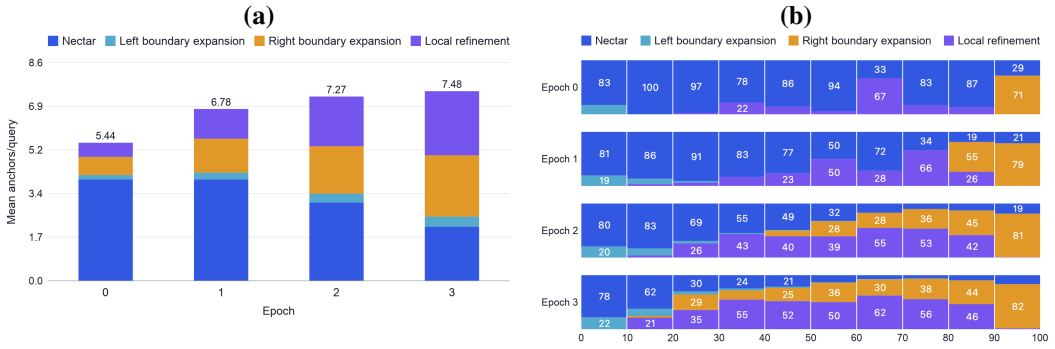


Figure 6: Dynamic buffer evolution: **(a)** anchor counts by source; **(b)** source composition along the quality scale.

4.5 Ablation Studies

Number of initial anchors. Table 4 varies the initial buffer size $K_0 \in \{2, 4, 7\}$ for both Nectar and bootstrap initialization. With Nectar initialization, $K_0 = 4$ performs best, improving the average by 1.17 points over $K_0 = 2$ and by 0.87 points over $K_0 = 7$. The seven-anchor setting also consumes 28.4% more judge tokens than the four-anchor setting. With bootstrap initialization, the best result instead occurs at $K_0 = 7$, which reaches 57.24 but uses 20.8% more judge tokens than $K_0 = 4$. Thus, increasing the number of anchors is not uniformly beneficial. The exact anchor subsets are specified in Appendix H.

Table 4: Effect of the initial anchor count.

Init.	K_0	AE2	AH-v2	WB-v2	Writing	Avg.	Tokens (M)	Calls (K)
Nectar	2	80.99	30.00	46.73	66.69	56.10	79.3M	36.4K
	4	82.80	30.20	48.79	67.27	**57.27**	88.8M	36.9K
	7	81.06	29.60	48.02	66.92	56.40	114.1M	36.8K
Bootstrap	2	80.62	27.90	47.55	66.79	55.72	66.5M	28.0K
	4	79.50	28.40	47.31	66.15	55.34	76.5M	28.4K
	7	79.50	33.50	48.58	67.39	**57.24**	92.4M	28.7K

Removing local fine ranking. Starting from RankBuffer, we evaluate a *coarse-only* variant that performs coarse ranking but omits local fine ranking; rollouts assigned to the same interval receive the same average rank. Table 5 shows that this variant reduces judge-token use by 16.5% and judge calls by 15.9%, but lowers all four benchmark scores and reduces the average from 57.27 to 54.34.

Table 5: Coarse-only ablation.

Variant	AE2	AH-v2	WB-v2	Writing	Avg.	Tokens (M)	Calls (K)
RankBuffer	**82.80**	**30.20**	**48.79**	**67.27**	**57.27**	88.8M	36.9K
Coarse-only	78.30	27.60	45.56	65.89	54.34	74.1M	31.0K

Removing anchor content. We additionally evaluate an *order-only* variant in which the coarse judge sees only the ordered anchor identifiers, rather than their response content, while local fine ranking remains unchanged. As shown in Table 6, AlpacaEval 2 drops from 82.80 to 51.93 with Nectar initialization and from 79.50 to 54.53 with bootstrap initialization. Thus, the anchor order alone is insufficient for placing new responses reliably.

Table 6: Order-only ablation.

Initialization	Variant	AlpacaEval 2
Nectar	RankBuffer	**82.80**
	Order-only	51.93
Bootstrap	RankBuffer	**79.50**
	Order-only	54.53

5 Conclusion

We introduced RankBuffer, a buffer-assisted framework for constructing relative rewards in open-ended group-based RL. RankBuffer reuses previously judged responses as an ordered, query-specific quality scale, combining parallel coarse ranking with selective local fine ranking to obtain a complete rollout order. Boundary expansion, local refinement, and inactive-anchor pruning adapt this scale as the policy evolves. Across four benchmarks, RankBuffer consistently outperforms all pointwise baselines and achieves performance comparable to the strongest ranking-based reward methods, while substantially reducing judging cost. The ablations demonstrate that local fine ranking improves downstream performance and that anchor response content is necessary for reliable coarse ranking. Buffer analyses further show that rollout-derived anchors progressively extend the covered quality range and increase ranking resolution in regions where current rollouts concentrate. Overall, RankBuffer provides an effective and efficient approach to relative reward construction by reusing prior judgments without requiring a separately trained reward model.

AI USE STATEMENT

We used generative AI tools only to aid and polish the writing of this paper, including improving grammar, clarity, and presentation. All AI-assisted edits were reviewed and revised by the authors, who take full responsibility for the final content.

ETHICS STATEMENT

Our study relies exclusively on publicly accessible datasets and involves no collection of personally identifiable information. Nevertheless, policies optimized with feedback from LLM judges may inherit limitations of the judge models, including systematic biases. The resulting models may also generate inaccurate information, unsubstantiated statements, or biased responses. These risks should be carefully considered when deploying the proposed framework in practical scenarios, particularly in high-stakes domains where such errors may have harmful consequences. For these applications, we encourage additional safeguards, such as factuality assessment, source and citation validation, and appropriate human review.

REPRODUCIBILITY STATEMENT

The main text describes the RankBuffer method, experimental setup, and evaluation protocol. The appendix provides complete pseudocode, judging prompts, implementation settings, baseline details, and ablation configurations. We also report the complete numerical results to facilitate verification and reproduction of our experiments. An anonymous repository containing the code and reproduction instructions is linked in the abstract.

REFERENCES

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Xiusi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru Wang, Yu Zhang, Denghui Zhang, Tong Zhang, et al. Rm-r1: Reward modeling as reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 88313–88342, 2026.

Yiqun Chen, Qi Liu, Yi Zhang, Weiwei Sun, Xinyu Ma, Wei Yang, Daiting Shi, Jiaxin Mao, and Dawei Yin. Tourrank: Utilizing large language models for documents ranking with a tournament-inspired strategy. In *Proceedings of the ACM on Web Conference 2025*, pp. 1638–1652, 2025.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. Gpg: A simple and strong reinforcement learning baseline for model reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 59637–59659, 2026.

Hongxin Ding, Baixiang Huang, Yue Fang, Weibin Liao, Zheng Li, Jinyang Zhang, Zhijing Wu, Junfeng Zhao, and Yasha Wang. Evorubrics: Dynamic rubrics as rewards via adversarial co-evolution for llm reinforcement learning. *arXiv preprint arXiv:2606.23038*, 2026.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *International Conference on Learning Representations*, volume 2026, pp. 127924–127945, 2026.

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

Zenan Huang, Yihong Zhuang, Guoshan Lu, Zeyu Qin, Haokai Xu, Tianyu Zhao, Ru Peng, Jiaqi Hu, Zhanming Shen, Xiaomeng Hu, et al. Reinforcement learning with rubric anchors. *arXiv preprint arXiv:2508.12790*, 2025.

Ruipeng Jia, Yunyi Yang, Yuxin Wu, Yongbo Gai, Siyuan Tao, Mengyu Zhou, Jianhe Lin, Xiaoxi Jiang, and Guanjun Jiang. Open rubric system: Scaling reinforcement learning with pairwise adaptive rubric. *arXiv preprint arXiv:2602.14069*, 2026.

Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 4334–4353, 2024.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *arXiv preprint arXiv:2406.11939*, 2024.

Bill Yuchen Lin, Yuntian Deng, Khyathi Chandu, Abhilasha Ravichander, Valentina Pyatkin, Nouha Dziri, Ronan Le Bras, and Yejin Choi. Wildbench: Benchmarking llms with challenging tasks from real users in the wild. In *International Conference on Learning Representations*, volume 2025, pp. 47852–47870, 2025.

Qi Liu, Bo Wang, Nan Wang, and Jiaxin Mao. Leveraging passage embeddings for efficient listwise reranking with large language models. In *Proceedings of the ACM on Web Conference 2025*, pp. 4274–4283, 2025a.

Qi Liu, Yanzhao Zhang, Mingxin Li, Dingkun Long, Pengjun Xie, and Jiaxin Mao. E2rank: Your text embedding can also be an effective and efficient listwise reranker. *arXiv preprint arXiv:2510.22733*, 2025b.

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, et al. Gdpo: Group reward-decoupled normalization policy optimization for multi-reward rl optimization. *arXiv preprint arXiv:2601.05242*, 2026.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 2511–2522, 2023.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective, 2025. *URL https://arxiv.org/abs/2503.20783*, 1, 2024.

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*, 2025c.

Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Morrison, Noah Smith, Hanna Hajishirzi, and Nathan Lambert. Rewardbench 2: Advancing reward model evaluation. In *International Conference on Learning Representations*, volume 2026, pp. 144839–144866, 2026.

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

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, et al. Large language models are effective text rankers with pairwise ranking prompting. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 1504–1518, 2024.

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

William F Shen, Xinchi Qiu, Chenxi Whitehouse, Lisa Alazraki, Shashwat Goel, Francesco Barbieri, Timon Willi, Akhil Mathur, and Ilias Leontiadis. Rethinking rubric generation for improving llm judge and reward modeling for open-ended tasks. *arXiv preprint arXiv:2602.05125*, 2026.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.

Chenglong Wang, Ziming Zhu, Yifu Huo, Bei Li, Qiaozhi He, Yan Ding, Xiaoyang Hao, Yuxin Gao, Tianhua Zhou, Xiaojia Chang, et al. Rrc: Unlocking generative reward models in llm reinforcement learning via ranking-based reward construction. *arXiv preprint arXiv:2608.06310*, 2026.

Xiyu Wei, Qingwei Zong, Xiaoguang Li, Eugene Yu, and Sujian Li. Qurl: Rubrics as judge for open-ended question answering. In *International Conference on Learning Representations*, volume 2026, pp. 107211–107233, 2026.

Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, et al. Writingbench: A comprehensive benchmark for generative writing. *Advances in Neural Information Processing Systems*, 38, 2026.

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

Ran Xu, Tianci Liu, Zihan Dong, Tony Yu, Ilgee Hong, Carl Yang, Linjun Zhang, Tao Zhao, and Haoyu Wang. Alternating reinforcement learning for rubric-based reward modeling in non-verifiable llm post-training. *arXiv preprint arXiv:2602.01511*, 2026b.

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

Wei Yang, Shixuan Li, Heng Ping, Peiyu Zhang, Paul Bogdan, and Jesse Thomason. Auditing multi-agent llm reasoning trees outperforms majority vote and llm-as-judge. *arXiv preprint arXiv:2602.09341*, 2026a.

Zixuan Yang, Yiqun Chen, Wei Yang, Erhan Zhang, Zihan Shen, Xiaochi Wei, Yan Gao, Yi Wu, Yao Hu, and Jiaxin Mao. Tournament-grpo: Group-wise tournament rewards for reinforcement learning in open-ended long-form generation. *arXiv preprint arXiv:2605.26958*, 2026b.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale, 2025. *URL https://arxiv. org/abs/2503.14476*, 1(2), 2025.

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

Erhan Zhang, Yiqun Chen, Zechun Niu, Wei Yang, Xiaochi Wei, Yan Gao, Yi Wu, Yao Hu, and Jiaxin Mao. Oases: Outcome-aligned search-evaluation co-training for agentic search. *arXiv preprint arXiv:2604.03675*, 2026a.

Qiang Zhang, Boli Chen, Fanrui Zhang, Ruixue Ding, Shihang Wang, Qiuchen Wang, Yinfeng Huang, Haonan Zhang, Rongxiang Zhu, Pengyong Wang, et al. Arenarl: Scaling rl for open-ended agents via tournament-based relative ranking. *arXiv preprint arXiv:2601.06487*, 2026b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Banghua Zhu, Evan Frick, Tianhao Wu, Hanlin Zhu, Karthik Ganesan, Wei-Lin Chiang, Jian Zhang, and Jiantao Jiao. Starling-7b: Improving helpfulness and harmlessness with rlaif. In *First Conference on Language Modeling*, 2024.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. Judgelm: Fine-tuned large language models are scalable judges. In *International Conference on Learning Representations*, volume 2025, pp. 51257–51296, 2025.

A RankBuffer Pseudocode

Algorithm 1 constructs rewards for one query group; Algorithm 2 maintains its ordered buffer. At each training step, Algorithm 1 is applied to each query in the batch, and the resulting advantages are used in the standard GRPO policy update. Each anchor's inactivity counter is initialized to zero when the anchor is created. Buffers and maintenance counters persist throughout training.

Algorithm 1 RankBuffer: group ranking and reward construction

Require: Query x, rubrics $\mathcal{R}_x$, rollouts $\mathcal{Y}_x^t$ of size G, epoch e
Require: Persistent buffer $\mathcal{A}_x$, initialization source, K_0, judge J
Require: Format coefficient λ_{fmt}, normalization constant ϵ, pruning threshold H
Ensure: Rewards r_i, advantages $\widehat{A}_i$, updated buffer $\mathcal{A}_x$

if $\mathcal{A}_x$ is uninitialized and the source is Nectar **then**
  $\mathcal{A}_x \leftarrow \text{Select}_{K_0}$(Nectar responses for x, worst to best)
  Initialize anchor inactivity counters and per-query epoch markers
end if
if $\mathcal{A}_x$ is uninitialized **then** ▷ Bootstrap: first-epoch query group
  $\sigma \leftarrow J(x, \mathcal{R}_x, \mathcal{Y}_x^t)$ ▷ Listwise, worst to best
  $\mathcal{A}_x \leftarrow \text{Select}_{K_0}(\sigma)$
  Initialize anchor inactivity counters and per-query epoch markers
else
  $K \leftarrow |\mathcal{A}_x|$; initialize empty buckets $\mathcal{C}_0, \ldots, \mathcal{C}_K$
  for all $y_i^t \in \mathcal{Y}_x^t$ in parallel **do**
    $p_i \leftarrow J(x, \mathcal{R}_x, \mathcal{A}_x, y_i^t) \in \{0, \ldots, K\}$
    Assign candidate i to $\mathcal{C}_{p_i}$
  end for
  for $p = 0, \ldots, K$ **do**
    if $|\mathcal{C}_p| \geq 2$ **then**
      $\sigma_p \leftarrow J(x, \mathcal{R}_x, \mathcal{C}_p)$ ▷ Listwise, worst to best
    else
      $\sigma_p \leftarrow \mathcal{C}_p$ ▷ Empty or singleton: no judge call
    end if
  end for
  $\sigma \leftarrow \sigma_0 \,\|\, \cdots \,\|\, \sigma_K$ ▷ Concatenate in interval order
  $\mathcal{A}_x \leftarrow \text{UpdateBuffer}(\mathcal{A}_x, \{\sigma_p\}_{p=0}^K, e, H)$
end if
for $i = 1, \ldots, G$ **do**
  $q_i \leftarrow$ zero-based position of candidate i in σ
  $r_i \leftarrow -1 + 2q_i/(G-1) + \lambda_{\text{fmt}} r_i^{\text{fmt}}$
end for
$\bar{r} \leftarrow G^{-1} \sum_{i=1}^G r_i$; $s \leftarrow \sqrt{(G-1)^{-1} \sum_{i=1}^G (r_i - \bar{r})^2}$
$\widehat{A}_i \leftarrow (r_i - \bar{r})/(s + \epsilon)$ for $i = 1, \ldots, G$
return $\{r_i, \widehat{A}_i\}_{i=1}^G, \mathcal{A}_x$

Here, Select_{K_0} uniformly subsamples an ordered sequence while retaining both endpoints; the experimental subsets are listed in Appendix H. Bootstrap initialization uses the first group ranking both to compute rewards and to select anchors, without an additional maintenance update on that group. In Algorithm 2, temporary coordinates refer to the pre-update buffer; additions and pruning are merged simultaneously so that pruning does not shift insertion intervals. New anchors have inactivity counter zero.

B Coarse- and Fine-Ranking Prompts

The following templates reproduce the instructions used for coarse ranking and local fine ranking, with runtime values replaced by placeholders. In both prompts, `WEIGHTED_RUBRICS` lists rubrics

Algorithm 2 Rule-based buffer maintenance

function UPDATEBUFFER($\mathcal{A}_x = (a_1, \ldots, a_K), \{\sigma_p\}_{p=0}^{K}, e, H$)
  $P \leftarrow \varnothing$ ▷ Indices of anchors to prune
  if inactivity has not yet been observed for query x in epoch e **then**
    for $k = 2, \ldots, K-1$ **do**
      if $|\sigma_{k-1}| + |\sigma_k| > 0$ **then**
        $h_k \leftarrow 0$
      else
        $h_k \leftarrow h_k + 1$
      end if
      if $h_k \geq H$ **then**
        $P \leftarrow P \cup \{k\}$
      end if
    end for
    Mark inactivity as observed for query x in epoch e
  end if
  $M \leftarrow \{(k, a_k) : 1 \leq k \leq K,\ k \notin P\}$
  if $|\sigma_0| > 0$ and the left boundary has not expanded in epoch e **then**
    Add $(\frac{1}{2}, \sigma_0[1])$ to M ▷ Worst rollout below the buffer
    Mark the left boundary as expanded in epoch e
  end if
  if $|\sigma_K| > 0$ and the right boundary has not expanded in epoch e **then**
    Add $(K + \frac{1}{2}, \sigma_K[|\sigma_K|])$ to M ▷ Best rollout above the buffer
    Mark the right boundary as expanded in epoch e
  end if
  for $p = 1, \ldots, K-1$ **do**
    if $|\sigma_p| \geq 3$ **then**
      $j \leftarrow \lceil |\sigma_p|/2 \rceil$ ▷ One-based lower median
      Add $(p + \frac{1}{2}, \sigma_p[j])$ to M
    end if
  end for
  Initialize $h = 0$ for every newly added anchor
  return Anchors in M sorted by their temporary coordinates
end function

in the format `1.[weight=5] rubric text.` Anchors are indexed from 0 in the prompts: `A0` corresponds to a_1^t in the main text, and `A(K-1)` corresponds to $a_{K_t}^t$.

B.1 COARSE-RANKING PROMPT

Here, `K` is the current number of anchors and `ORDERED_ANCHOR_BLOCKS` contains each anchor ID followed by its response text, ordered from worst to best. The judge returns an insertion position rather than a pointwise reward.

```
You are a strict response-quality judge. The anchors below are
already ordered from WORST to BEST.
Place the candidate response into exactly one insertion position
using the query and weighted rubrics.

Position semantics for {K} anchors:
- 0: candidate is worse than A0
- i (1 <= i < {K}): candidate belongs between A(i-1) and Ai
- {K}: candidate is better than A{K-1}

QUERY
{QUERY}

RUBRICS
{WEIGHTED_RUBRICS}
```

```
ORDERED ANCHORS (worst to best)
{ORDERED_ANCHOR_BLOCKS}

CANDIDATE
{CANDIDATE_RESPONSE}

Return JSON only: {"position": <integer from 0 through {K}>}
```

B.2 Fine-Ranking Prompt

During fine ranking, this prompt is used for intervals containing at least two rollouts. `CANDIDATE_BLOCKS` contains all rollouts in that interval, each preceded by `[R<ID>]`. `CANDIDATE_IDS` is the list of their integer IDs. The returned `order` must contain each ID exactly once; anchors are not included in this judgment. Bootstrap initialization uses the same listwise prompt, with the initial rollout group as candidates and no coarse ranking.

```
You are a strict response-quality judge. Rank only the candidate
responses below from WORST to BEST.
Use the query and weighted rubrics. Every candidate ID must
occur exactly once.

QUERY
{QUERY}

RUBRICS
{WEIGHTED_RUBRICS}

CANDIDATES
{CANDIDATE_BLOCKS}

Return JSON only. `order` must be a permutation of
{CANDIDATE_IDS}, from worst to best:
{"order": {CANDIDATE_IDS}}
```

C Implementation Details

Training uses one node with eight NVIDIA H800 GPUs, fully sharded data parallelism for policy optimization, and vLLM for rollout generation. Table 7 summarizes the training, sampling, and buffer-maintenance settings. Unless otherwise specified, these settings are shared across experiments; the initial anchor source, buffer size, and maintenance protocol vary as described in the corresponding experiments.

D Rubric Generation

We generate query-specific rubrics using DeepSeek-V4-Flash (Xu et al., 2026a) before RL training. For each query, the model receives the query and its seven Nectar responses in best-to-worst order and generates three to five concise rubrics. Each rubric is assigned an integer importance weight from 1 to 5. We use a temperature of 0.2 and a maximum output length of 1,024 tokens. The generated output is parsed as JSON and validated to ensure that the number of rubrics and their weights satisfy the required constraints. All successfully generated rubrics are kept fixed throughout RL training.

The complete system and user prompts are provided below.

D.1 System Prompt

```
You are an expert evaluator for assistant responses.
Given one user query and seven responses that are already sorted
```

Table 7: Main implementation settings.

Setting	Value
Training	
Queries per training step	20
Rollouts per query (G)	8
Policy learning rate	1×10^{-6}
Maximum prompt length	4,096 tokens
Maximum response length	4,096 tokens
Format-reward coefficient (λ_{fmt})	0.2
Sampling and judging	
Policy temperature	0.6
Policy top-p	0.95
Policy top-k	20
Judge decoding	Deterministic
Rule-based buffer maintenance	
Default initial buffer size (K_0)	4
Interval-refinement threshold	$\lvert\mathcal{C}_p^t\rvert \geq 3$
Inactivity-pruning threshold (H)	3 epochs

```
from best to worst, write 3 to 5 concise judging rubrics that
explain the key dimensions that separate better answers from
worse ones for this specific query.

Requirements:
- The rubrics must be specific to this query and these seven
  responses.
- The rubrics should be useful for a later judge model to
  compare new responses.
- Focus on high-level evaluation dimensions such as correctness,
  completeness, instruction following, clarity, reasoning
  quality, safety, and style when relevant.
- Avoid mentioning the original seven answers explicitly in the
  rubric wording.
- Avoid redundant rubrics.
- Each rubric should be one sentence or short phrase.
- Assign each rubric an integer weight from 1 to 5.
- Larger weights mean the rubric is more important for judging
  future answers to this query.
- Use weight 5 only for the most critical dimensions.
- Output valid JSON only.

Return JSON in the following format:
{
  "rubrics": [
    {
      "text": "rubric 1",
      "weight": 5
    },
    {
      "text": "rubric 2",
      "weight": 4
    }
  ]
}
```

D.2 User Prompt

```
Below is one query and seven candidate answers that are already
ranked from best to worst.

Your task:
1. Infer what makes the higher-ranked answers better than the
   lower-ranked ones.
2. Write 3 to 5 rubrics that can later be used to judge newly
   generated answers for the same query.
3. Assign each rubric an importance weight from 1 to 5, where 5
   means most important.
4. The rubrics should be general enough to evaluate future
   answers, but grounded in the distinctions visible in these
   seven ranked answers.

Query:
{QUERY}

Seven ranked answers (best to worst):

[Rank 1 | model={MODEL_1}]
{ANSWER_1}

[Rank 2 | model={MODEL_2}]
{ANSWER_2}

[Rank 3 | model={MODEL_3}]
{ANSWER_3}

[Rank 4 | model={MODEL_4}]
{ANSWER_4}

[Rank 5 | model={MODEL_5}]
{ANSWER_5}

[Rank 6 | model={MODEL_6}]
{ANSWER_6}

[Rank 7 | model={MODEL_7}]
{ANSWER_7}
```

E Baseline Implementation Details

Pointwise reward methods. For GRPO, Dr. GRPO, DAPO and GPG, each valid rollout is independently scored once by the judge on a 0–10 scale using the query-specific weighted rubrics. The score s is mapped to $2s/10 - 1$ before adding the format reward. GDPO separately evaluates each rubric and normalizes the rubric-specific rewards before aggregation, resulting in more judge calls.

Tournament and RRC methods. Tournament-GRPO runs three independently shuffled binary tournaments for each group of eight rollouts, requiring 21 logical comparisons when all requests succeed. RRC-SCR compares every unordered pair in the group. RRC-AGR compares each rollout against the seven fixed Nectar responses for its query. This is a reproducible adaptation because the RRC paper specifies anchors sampled from a reference policy but does not fix a public reference checkpoint; Nectar ranks are not exposed to the AGR reward.

Exhaustive pairwise ranking. Pairwise Rank judges every unordered pair of non-empty responses; a non-empty response automatically wins against an empty one. We rank non-empty re-

sponses by decreasing win rate over completed comparisons. Equal win rates share the first rank occupied by the tied responses (e.g., $1, 1, 3$). For n non-empty responses, rank s_i (1 is best) is mapped to $2(n - s_i)/(n - 1) - 1$ for $n > 1$ and 0 for $n = 1$, before adding the format reward. Empty responses receive a semantic reward of -1.

ArenaRL. Our ArenaRL implementation uses the first rollout as a deterministic seed response, compares each sampled rollout against that seed, and then runs a seeded single-elimination tournament. Each logical match is judged in both candidate orders. A group of eight therefore requires 14 logical matches and 28 successful judge requests under error-free execution.

Across the ranking-based reward baselines, candidate presentation order is randomized with a deterministic hash of the global seed, query, training step, and candidate identifiers. Malformed or failed judge responses are retried; only successfully parsed requests contribute to the reported call and token totals.

F Complete Numerical Comparison

Table 8 reports the values underlying Figures 4 and 5. Judge tokens count prompt and completion tokens used for reward construction during training; calls count successfully parsed judge requests. The four benchmark columns use the controlled evaluation protocol described in Section 4.1.

Table 8: Complete effectiveness and judge-cost comparison. An asterisk (*) denotes static buffers.

Method	AE2	AH-v2	WB-v2	Writing	Avg.	Tokens	Calls
GRPO	72.73	27.50	46.57	65.42	53.06	25,035,966	31,374
GDPO	64.35	20.10	45.14	64.94	48.63	86,937,684	144,236
Dr. GRPO	56.15	18.90	44.32	63.37	45.60	21,936,802	31,649
DAPO	74.22	26.60	46.83	66.13	53.44	24,078,025	31,231
GPG	53.04	16.90	45.78	64.98	45.18	20,915,602	31,619
Tournament-GRPO	80.75	29.50	48.87	68.16	56.82	157,313,284	84,000
RRC-AGR	71.74	23.30	46.61	64.79	51.61	200,817,235	218,914
RRC-SCR	82.92	29.80	49.41	68.70	57.71	200,817,235	104,028
Pairwise Rank	81.80	27.00	47.88	68.83	56.38	188,264,527	103,035
ArenaRL	82.24	30.30	49.87	67.48	57.47	154,935,508	101,388
RankBuffer	82.80	30.20	48.79	67.27	57.27	88,818,250	36,861
RankBuffer*	79.69	26.70	46.95	66.71	55.01	67,967,890	36,227
RankBuffer-Bootstrap	79.50	28.40	47.31	66.15	55.34	76,454,821	28,378
RankBuffer-Bootstrap*	81.61	29.60	47.47	67.48	56.54	60,662,185	28,108
LLM	81.74	30.50	47.34	66.83	56.60	96,503,178	41,592
LLM + look-ahead	79.07	28.40	46.31	66.13	54.98	110,997,178	42,101

G LLM Buffer Variants

The LLM variant starts from the same four Nectar anchors as RankBuffer. After the current group has been ranked, the manager receives the query, weighted rubrics, four current anchors, current rollouts, and their combined worst-to-best ordering. It must select exactly four responses from the union of old anchors and current rollouts. The selected responses are copied verbatim and sorted according to the established order to form the next buffer. Invalid outputs leave the buffer unchanged.

The LLM + look-ahead variant delays this update until the query is encountered in the next epoch. The manager additionally sees that epoch's newly sampled, unranked rollouts as context, but may select only from the old anchors and the previous epoch's ranked candidates. After selection, the resulting buffer is used to rank the new rollouts.

The core selection instruction is:

```
Select the members of the next fixed-size reference buffer.
The buffer is a quality scale, not a collection of only the best
answers. Keep a stable, non-redundant set that covers the
response quality spectrum and meaningful transitions under the
rubrics.
Prefer a rollout when it fills a gap or is a better
representative than a current anchor.

Return valid JSON with exactly four selected IDs:
{
  "keep_anchor_ids": [IDs of retained current anchors],
  "add_rollout_ids": [IDs of selected eligible rollouts]
}
The two lists must contain four IDs in total. IDs not returned
are removed or discarded. Do not select look-ahead rollouts.
```

H Ablation Configuration Details

Initial-anchor subsets. For Nectar initialization, the seven responses are first ordered from worst to best and then uniformly subsampled while preserving both endpoints. The selected Nectar ranks are $(7, 1)$ for $K_0 = 2$, $(7, 5, 3, 1)$ for $K_0 = 4$, and $(7, 6, 5, 4, 3, 2, 1)$ for $K_0 = 7$. For bootstrap initialization, we first rank the eight rollouts sampled for each query in the first epoch from worst to best and apply the same endpoint-preserving uniform subsampling. The selected positions in that ordered group are $(1, 8)$ for $K_0 = 2$, $(1, 3, 6, 8)$ for $K_0 = 4$, and $(1, 2, 3, 5, 6, 7, 8)$ for $K_0 = 7$. Subsequent buffer expansion, refinement, and pruning are unchanged across the six variants.

Order-only configuration and judge prompt. The order-only control keeps the query, weighted rubrics, candidate response, and the existing worst-to-best sequence of anchor identifiers, but removes the anchor response text. Local fine ranking is otherwise unchanged. Its coarse-ranking prompt is:

```
You are a strict response-quality judge. The anchors below are
already ordered from WORST to BEST. Place the candidate response
into exactly one insertion position using the query and weighted
rubrics.

Position semantics for K anchors:
- 0: candidate is worse than A0
- i (1 <= i < K): candidate belongs between A(i-1) and Ai
- K: candidate is better than A(K-1)

QUERY
{QUERY}

RUBRICS
{WEIGHTED_RUBRICS}

ORDERED ANCHORS (worst to best)
[A0]
...
[A(K-1)]

CANDIDATE
{CANDIDATE_RESPONSE}

Return JSON only: {"position": <integer from 0 through K>}
```